CRANFIELD INSTITUTE OF TECHNOLOGY

CENTRE FOR LOGISTICS AND TRANSPORTATION

SCHOOL OF MANAGEMENT

PhD THESIS

Academic Year 1991-92

A. E. S. C. BRITO

Configuring Simulation Models Using CAD Techniques: A New Approach to Warehouse Design

Supervisor: R Saw

October 1992

ABSTRACT

The research reported in this thesis is related to the development and use of software tools for supporting warehouse design and management. Computer Aided Design and Simulation techniques are used to develop a software system that forms the basis of a Decision Support System for warehouse design.

The current position of simulation software is reviewed. It is investigated how appropriate current simulation software is for warehouse modelling. Special attention is given to Visual Interactive Simulation, graphics, animation and user interfaces.

The warehouse design process is described and common difficulties are highlighted. A Decision Support System (DSS) framework is proposed to give support during all the warehouse design phases.

The use of simulation in warehouse design is identified as being essential for evaluating different warehouse configurations. Several simulation models are used to show that the warehouse systems special characteristics require a new way of defining the simulation model and new modelling elements to represent the complex logic of a warehouse system.

AWARD (Advanced WARehouse Design) is a data-driven generic model, developed to build warehouse simulation models. It uses Computer Aided Design (CAD) techniques for drawing the warehouse layout and configuring the simulation model. The user has no need for programming skills and a user-friendly interface makes it easy to use. High resolution colour graphics and a scale drawing of the warehouse makes the dynamic display of the model a good representation of the real system.

Several examples illustrate the use of the AWARD system. The experience and advantages of the AWARD approach is discussed and the extension of this approach to other areas is explored.

ACKNOWLEDGEMENTS

I offer my thanks to all the companies from the AWARD consortium for their interest and participation and also for their technical and financial support.

I also thank to everyone in the Distribution Studies Unit (DSU) from the Cranfield Institute of Technology for their support during all these years.

I am grateful to all my colleagues in the Department of Mechanical Engineering and Industrial Management from the University of Oporto, Portugal, specially to those working at GEIN, for their continuous support and encouragement.

To everyone from the Portuguese and Brazilian community at Cranfield and specially to my friends Paulo Guedes and Fernando Silva my deepest thanks.

My special thanks to my colleague and friend J. A. Barros Basto for his dedication and contribution to this work and for the energy that he has always shown during the long working hours.

I am very grateful to Prof. Carlos Moreira da Silva for his initial guidance and support; without those this work could never be done.

I am also very grateful to my supervisor Mr. Richard Saw for helping me in many ways and for his guidance and support.

To my Parents for everything they have done during my life, my gratitude.

Finally to Graça, Diogo and Bárbara an apology for the time not spent with them and my deepest gratitude for their love and understanding.

CONTENTS

FIGURES

CHAPTER 1

INTRODUCTION TO SIMULATION

1. Introduction to Simulation

1.1. Introduction

Simulation is a generic term which is often used to describe very different types of activities. There are many definitions in the literature for simulation. We choose the following one from Shannon [Shannon 1975]:

> "Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behaviour of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system."

The use of computers for building and running simulations models, although not mandatory, has become common. In this case we refer to simulation as computer or digital simulation (assuming that a digital computer is used). Also, simulation is used today in numerous areas but our main concern, will be the simulation as applied in operational research.

There are different ways of classifying simulation models based on their characteristics. The most common are deterministic vs. stochastic and discrete vs. continuous. A deterministic model is one where we can predict its behaviour. So any state of the model can be defined by its previous state and by the activity that caused the change of state. In a stochastic model that is not true, because there is a certain amount of randomness involved. Though it is usually possible to know how likely the model is going to behave.

The terms discrete and continuous refer to the way a model is changing its state with time; if the changes in the model state are continuous over time then we have a continuous model; if the changes in the model state happens in a discrete way then we have a discrete model. Next, is considered the history and developments in simulation.

1.2. History and Developments in Computer Simulation

Computer simulation has become widely used in the analysis of complex problems and as an important tool in decision making. Since the 1960s simulation has been used with success in solving real problems. The developments in computer simulation have been closely related with the developments in computer hardware and software. Some years ago the computer costs of running simulation models were so high that often, the use of this technique was just not considered at all.

On the other hand, programming was made at a very low level and usually machine dependent and so good programming skills were needed for implementing the simulation models. The increase in computer capacities and performance, the reduction in price and maintenance costs together with the generalized use of microcomputers and more friendly user interfaces are some of the reasons that explain why simulation techniques are now so commonly used. Pidd [Pidd 1984] gives a good description about the way computers had influenced the developments in simulation.

Simulation has been used for a long time now. The first book about computer simulation was from Tocher [Tocher 1963] and was written in the early 1960's. The first simulation models using a computer had to be implemented in machine code or assembly. This of course had a lot of disadvantages: the model logic had to be simple; the testing and debugging was very difficult and time consuming; the models were hardware dependent so could not be used in different computers; the computer costs were very high.

With the introduction of programming languages like FORTRAN, ALGOL and COBOL the task of programming simulation models became less difficult. In addition to that the source code could be ported between different computers, providing there was a compiler available, so the work previously done in other computers could be used again. The use of high level languages led people to write programs in a more structured way, building subroutines and procedures for different tasks in a simulation model.

The next logical step was organizing sets of simulation subroutines that were common to different simulation models. People realize that a lot of functions within the simulation models could be made generic and included in subroutine libraries for future use in other similar jobs. Examples of this libraries for discrete simulation are the

ALGOL procedures of SIMON [Hills 1965] and the FORTRAN subroutines of GASP [Pritsker 1974].

The use of general purpose programming languages for building simulation models have some disadvantages. The most important ones are the fact that in a programming language there are no elements that can translate the simulation concepts and there is no error checking that can help the user in debugging the model. So paralleling to the developing of simulation libraries, during the 1960s, there was also, a development in what was called simulation languages. These languages were created with the only objective of building simulation models. This approach has obvious benefits mainly for the user not familiar with programming languages because he can learn simulation and build models with a much higher level of abstraction.

One of the first examples of a simulation language was developed by Esso and was called Control and Simulation Language CSL, [Buxton & Laski 1962]. In CSL the simulation source code was transformed into FORTRAN source code that then could be compiled and run on the machine. Another language that used a similar approach was SIMSCRIPT [Markowitz et al. 1963] developed at the RAND Corporation in USA. SIMSCRIPT instructions were an extension to the FORTRAN instructions set with a syntax suitable for building simulation models. The SIMSCRIPT pre-processor translates the simulation instructions to FORTRAN that then could be compiled and run as in CSL.

For developing simulation models using subroutine libraries, as SIMON or GASP, or using a simulation language, like CSL or SIMSCRIPT, the analyst needs to have some programming experience. Mostly the potential simulation user was not very familiar with computers and had little programming expertise. So it was natural that people started to search for other ways of defining simulation models with more user friendly interfaces and with a less need for computer programming skills.

One of the solutions adopted was the utilization of block diagrams. Using this principle the task of defining a model consists of choosing the right blocks and putting them in the right order so that the block diagram obtained represents the system to be simulated. An example of the use of block diagrams is the General Purpose System Simulator, GPSS [Greenberg 1972] and [Gordon 1979], that had origin, in the beginning of 1960s, at the Bell Telephone Laboratory with the support from IBM.

From the middle 1960s to the middle 1970s there was an evolution of the simulation languages. The CSL language was extended and became the Extended Control and Simulation Language ECSL [Clementson 1982]. SIMSCRIPT a very popular language, mainly in USA, was updated to SIMSCRIPT 2.5 [Kiviat et al. 1973].

There was also, other simulation languages that appeared during this period such as SIMULA [Hills 1973], developed at the Norwegian Computing Centre, that is an extension of the programming language ALGOL 60. Improvements were also made to simulation subroutines libraries like GASP, that became GASP IV, and to block diagrams systems like GPSS which went through several versions.

In 1976 the concept of Visual Interactive Simulation was introduced by Hurrion [Hurrion 1976]. With VIS a picture of the model running was displayed on the screen and the user could interrupt the model running, at any time, and interact with it in a way that he could influence the future behaviour of the model. The mentioned advantages of using VIS were: the model validation was easier; the interaction with the model increased the confidence in it; the visual aspects helped to detect problems and to involve managers in the model development.

In 1979 there become available the first VIS commercial package SEE-WHY, [Fiddy et al. 1981], developed at the Operational Research group at British Leyland. The SEE-WHY models were developed in FORTRAN using the available SEE-WHY subroutines. Another package, similar to SEE-WHY, was released in 1981 by the OR group of the British Steel Corporation and was called FORSSIGHT [Hollocks 1983b] and later marketed in the USA as WITNESS. This package had improved visual facilities as icons and a separate interactive program to generate graphic displays and icons.

Since the early 1980's there has been a large increase in the use of microcomputers and they have been used for simulation purposes as well. The fact that microcomputers tend to be easy to use and have colour and graphics capabilities, without being very expensive, led microcomputer software to become more user friendly and have high quality displays and graphics.

Most of the simulation packages adapted their software to run on PCs, specially on IBM-PC compatibles. A lot of effort has been made since then to make easier the task of building simulation models and to improve the visual aspects of simulation at different

levels. Colours and graphics have been used for defining and running the model and also for making the analysis of the results.

Another way of trying to simplify the analyst's task are simulation program generators. They accept a simple definition of the model and generate the complete source code that can be executed as it is, or modified by the user. Since the 1970s [Mathewson 1974] simulation program generators have been implemented and used. Because this type of program is more suitable to be used in an interactive environment they usually are interactive program generators.

Examples of these are DRAFT [Mathewson 1977] that can generate code in ALGOL, FORTRAN and SIMULA and CAPS [Clementson 1982] which generate ECSL code. DRAFT has a recent version [Mathewson 1985] that produces FORTRAN code and use graphics animation on an IBM PC compatible.

Although reducing the lead time necessary to build a model, the use of ISPG's demand a certain experience in simulation modelling. A different approach which attempt to overcame that is what Pidd [Pidd 1988] calls Data-Driven Generic Models. The idea is to have a generic model for a class of problems and each particular model within that class is defined through data specified to the generic model. This has of course the disadvantage of limited scope but it has the great advantages of the reduced time in defining the model and the low level of expertise required.

More recently there has been some work done in creating a set of software tools to help the user in the different stages of solving a problem using simulation. These software tools form what can be called an integrated environment for simulation. An example of this is the Computer Aided Simulation Modelling CASM [Balmer & Paul 1986] a project at the London School of Economics for automating the task of simulation modelling. The use of data bases, Artificial Intelligence and Expert Systems are also assuming an important role in the development of this integrated environment.

In general, we can say that special attention has been given to visual aspects of simulation, in particular to graphics and animation. Different kind of development tools are being produced to help the user in the creation, validation and analysis of the results of simulation models in a way that more people can use this powerful technique.

1.3. Objectives of This Work

Most of the inspiration and incentive for this work came from many discussions established with different persons from the warehouse industries. Without this it would have been difficult to identify some of the problems involved with warehouse design and management. Also, these contacts were very important in the evaluation of the solutions that were being developed.

The main concern of this work is related to the development and use of software tools for supporting warehouse design and management. Computer Aided Design and Simulation techniques are used to develop a software system that forms the basis of a Decision Support System for warehouse design. Hence, the objectives of this work were as follows.

The first objective was to review the current position of simulation software. This is a difficult task as simulation software has been improved rapidly in the past few years. Special attention was given to Visual Interactive Simulation software as it is our belief that this approach has many advantages over other simulation approaches.

The second objective was to investigate how appropriate current simulation software is for warehouse modelling. This objective includes not only the analysis of the simulation software suitability for developing warehouse models, but also to determine whether the way the model is defined is adequate for warehouse designers.

The third objective was to develop the most appropriate simulation environment for warehouse design and study purposes, taking into account performance, reporting and the user interface.

In summary, this work is seen as examining the way software is used now for a particular purpose and attempting to use modern interactive graphic techniques to improve the application.

CHAPTER 2

SIMULATION CONCEPTS

AND

LANGUAGES

2. Simulation Concepts and Languages

2.1. Discrete Simulation Concepts

In the definition of simulation, given in 1.1., the words model and system were used. Although these words are quite common it is important to clarify their meaning. Shannon [Shannon 1975] gives the following definitions:

> "A system is a group or set of objects united by some form of regular interaction or interdependence to perform a specified function."

> "A model is a representation of an object, system, or idea in some form other than that of the entity itself."

As it was said in 1.1. there are different classifications for models depending on their characteristics. Our main concern here will be the stochastic and discrete modelling. So, we are interested in systems that can be represented by a series of events, that happen discretely through time, and which behaviour presents some level of randomness. Although there are models that are time independent, usually called static models, normally the models are time dependent and are called dynamic models. In this case, when it comes to build the model, the user must choose a way to handle the simulation time flow. According to Pidd [Pidd 1988] there are two techniques for time handling:

Time slicing - the model is examined and updated at regular intervals. Thus, for a time slice of length dt, the model is updated at a time (t + dt) for changes occurring in the interval (t to (t + dt)).

Next event - The model is examined and updated when it is known that a state change is due. These state changes are called events. So time is changed from one event to the next. That is why this technique is called next event.

The next event technique has more advantages and is usually more efficient when compared with the time slicing technique. This is due mainly to the fact that in the next event technique the time increment adjusts itself to the frequency in which events occur. Because of the fixed length time interval in the time slicing technique, the model is

examined unnecessarily, as nothing has happened. Sometimes there is a loss in precision, as many things have happened. When the events occur at regular intervals then the use of the time slicing technique could be preferred due to its simpler concepts and easier implementation.

The terminology in discrete simulation is not standardized yet so it seems useful to give here some definitions:

Entities - Are simulation elements that can be individually identified and processed;

Classes - Are groups of like entities;

Attributes - Are a way of keeping information associated with an entity;

Queues - Are places where entities wait for something to happen; usually have an associated discipline such as first in first out FIFO;

Event - Is the instant of time corresponding to a change in the state of the system;

Activity - Is an operation or operations that change the state of an entity;

Process - Is a group of events which has a special meaning when they occur in their chronological order; a good example is the path of an entity through the system;

Simulation clock - Is a simulation element that has the current simulation time.

Another concept often used in discrete simulation is the activity cycle diagram. This diagram is a way of representing graphically the interactions between entities in a system. After identifying the entities in a system as well as the activities which involved them the user must draw an activity diagram that shows the history of each entity, or

class of entities, and the interactions between them. The two main symbols used in activity diagrams are shown in figure 2.1.

Figure 2.1 The two main symbols used in activity diagrams

Circles are used to represent queues and rectangles are used to represent activities. the activity cycle diagrams are good ways of describing the structure of a model but sometimes they are not capable of representing the full complexity of a system. Even so, they can be useful for defining the overall structure of the model that can be detailed afterwards. After having introduced the main concepts in discrete simulation, next it is presented the different approaches to modelling in discrete simulation.

2.2. Discrete Simulation Modelling

2.2.1. Program Structure

The program structure for discrete modelling is shared by the different approaches and is defined by Fishman [Fishman 1973] as having the following three level hierarchical structure:

Level 1: executive (control program);
Level 2: operations;
level 3: detailed routines.

The executive controls the sequence in which the operations at level 2 are executed. The next event technique is usually used to update the simulation time and select the operations due to occur at that time. At the level 2 we found the operations that reproduce the behaviour of the model. The way these operations are described depends

on the chosen modelling approach. The analyst can choose between the following four approaches:

event approach;
activity approach;
process interaction approach;
three phase approach.

Each one has its own executive and structure for level 2 that will be described later. The third level includes auxiliary modules used by level 2. These can consists of samples, reports, graphs, statistics, etc. Normally, the analyst should only be worried with building the model at level two. If a subroutine library or a simulation language is used they usually take care of the first and third levels. However the user must have a reasonable understanding of how they work to be able to define the logic at level two. It is also important that the user knows the principles behind the modelling approach used in order to implement the operations that describe the model.

2.2.2. Event Approach Modelling

In the event approach the operations at level two are implemented using the event as the basic building block. The executive controls the time advance and the order in which the events are executed. Figure 2.2 shows a diagram representing the logic for an event-based executive. The use of a time ordered list of events facilitate the time scan and event selection tasks of the executive. The simulation time is always advanced to the time the next event is due and the next event is the one at the top of the list. The selected event is removed from the events list and is executed. To each event is associated a routine which is called when that event is due. This approach is more common in USA than in Europe. A good example of a simulation language using the event-based approach is SIMSCRIPT [Markowitz et al. 1963].

2.2.3. Activity Approach Modelling

In the activity approach the basic building block for defining the operations at level two is the activity. An activity describes the actions that will follow a certain state change of

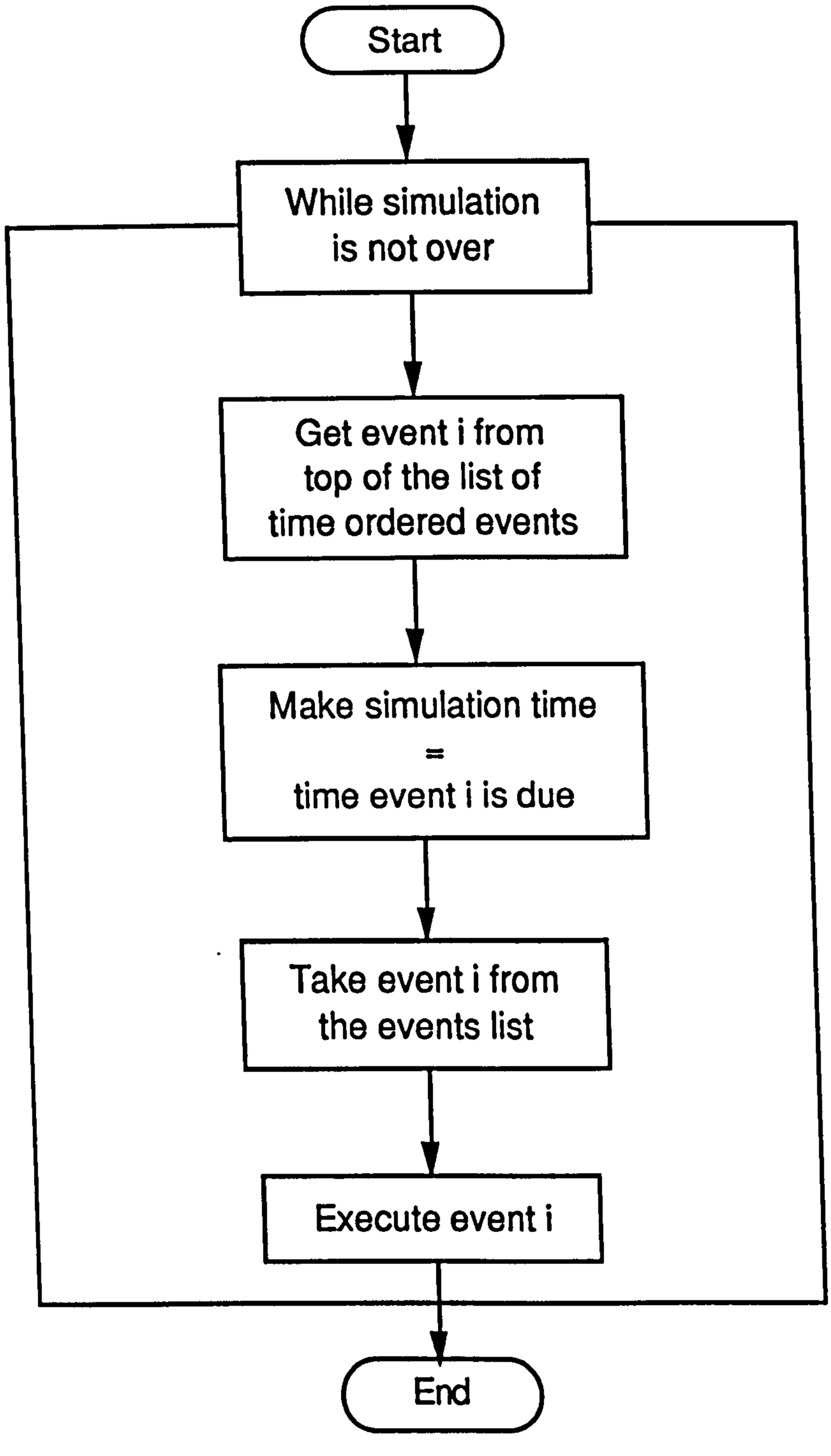


Figure 2.2 Logic diagram for an event-based executive

the system. These actions will only take place if the necessary conditions are met. So according to Pidd [Pidd 1988] the implementation of activities present the following two part structure:

test head: the conditions which must be satisfied if the activity is to be executed;

actions: the operations which constitute the activity; only performed if the tests are passed.

This structure is used to build the routines associated with each activity. As in the event approach the next event technique is used to advance the simulation clock. At each event the activities are scanned and the actions will be executed depending on the conditions at that time. In the activity approach the executive uses time cells, instead of an events list as in the event approach, to do the time scan in order to know when the next event is due. Time cells, as defined by Pidd [Pidd 1988], are attributes of the permanent entities that indicate when each entity is due to change state. Also, according to Pidd [Pidd 1988] there are the two following ways of doing the time scan task:

```
1 -  FOR all permanent entities with time cell > clock
          FIND minimum time cell
     clock = minimum time cell
```

```
2 -  FOR all permanent entities with time cell > 0
          FIND minimum time cell
     clock = clock + minimum time cell
     FOR all permanent entities
          time cell = time cell - minimum time cell
```

Special attention should be given to the order in which the activities are scanned because the model behaviour can become distorted. One way of overcoming this problem is to repeat the scan process until no action is performed. Figure 2.3 shows a logic diagram for an activity-based executive. The activity approach was quite popular in UK and was used by the simulation languages CSL [Buxton & Laski 1962] and ECSL [Clementson 1982].

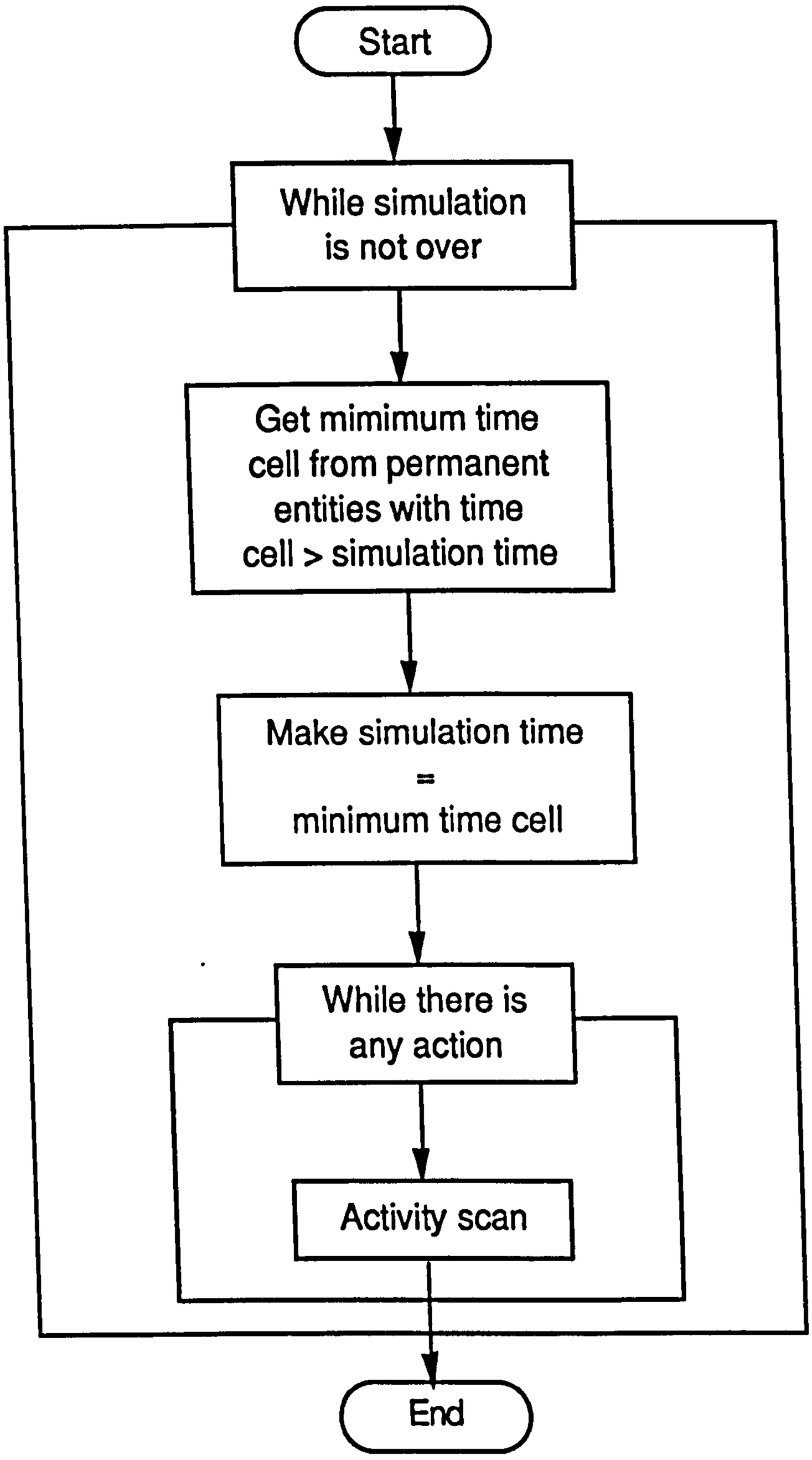


Figure 2.3 Logic diagram for an activity-based executive

Comparing the activity with the event approach it is easy to conclude that the event approach is much more efficient. The reason why, is that in the event approach the events go to the events list only if they can be executed. In the activity approach the

activities must be scanned and the conditions tested even if the activities cannot be executed. Although less efficient the activity approach offers some advantages against the event approach. The activities are much easier to implement because the analyst does not need to worry about their sequence of execution as in the event approach. That usually gives smaller program modules and the logic is easier to understand. The three phase approach [Tocher 1963], described later, has overcome some of the inconvenience of the activity approach.

2.2.4. Process Interaction Approach Modelling

The process interaction approach uses the process as the basic building block. As defined in 2.1 a process can be identified with the sequence of operations that describe the path of an entity through the system. The operations at level two are implemented normally using a separate routine for each different process. This approach seems to be the more intuitive and probably will be used by someone who wants to build a model and has no previous simulation experience. In the process interaction approach each entity will be followed through the set of operations corresponding to its process. The progress of an entity depends on certain conditions and on the state of the other process. The progress of an entity can be delayed by a known period of time, as in an operation in which it is involved, or by an unknown period of time that depends on how the system is going to move ahead. So the entity will move from one point to another along its process. These points are called [Pidd 1988] re-activation points as they mark locations in the process where the entity progress will be activated after being delayed.

The process interaction-based executive is much more complex than the event-based and activity-based executives. This conclusion can be easily taken comparing the diagram for a process interaction-based executive, from figure 2.4, with the diagrams from figures 2.2 and 2.3. To implement a process interaction executive it is useful to group the following information in a record [Pidd 1988]:

- the entity concerned;
- its next re-activation point.

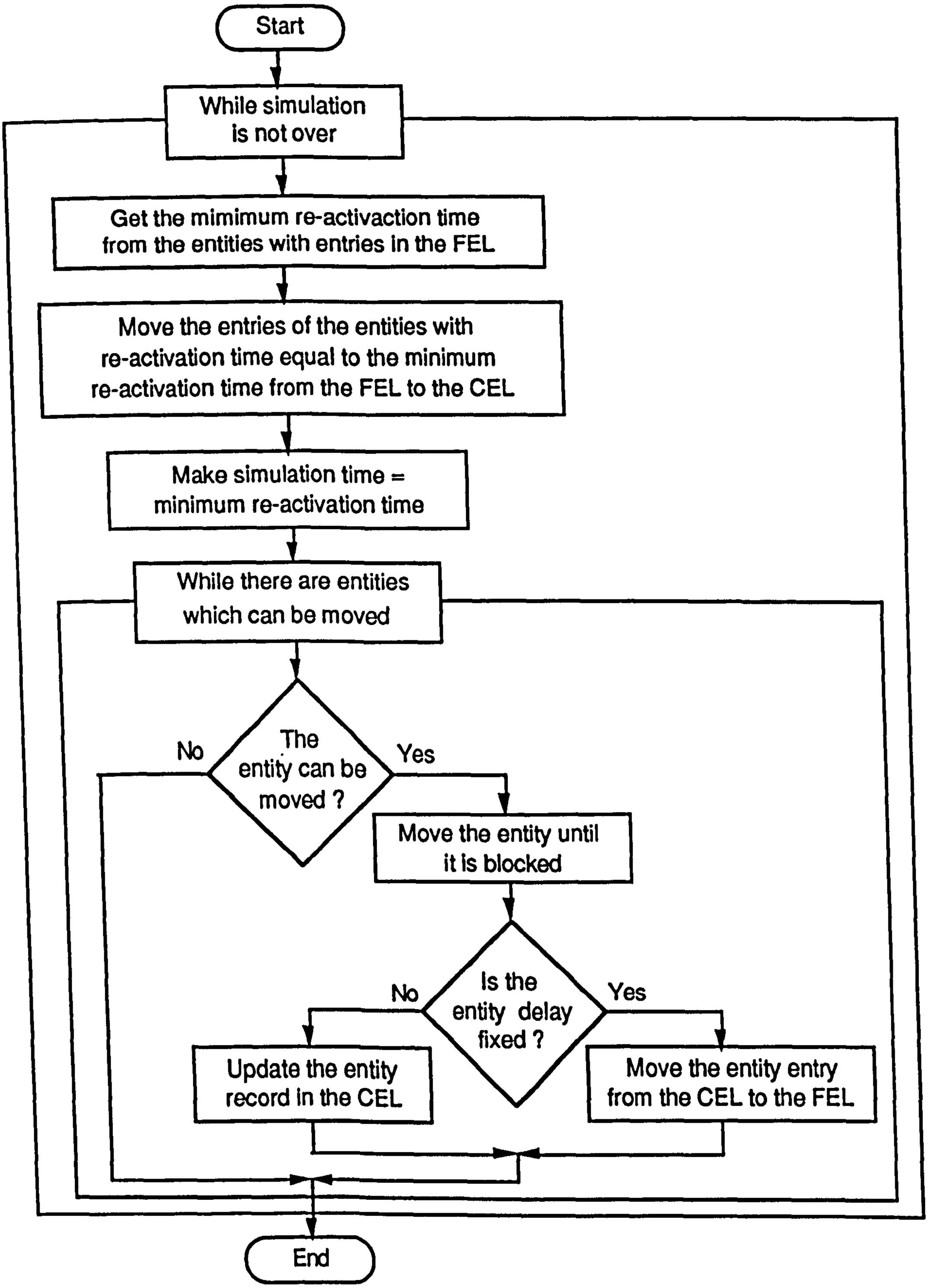


Figure 2.4 Logic diagram for a process interaction-based executive

These records are used to keep track of the entities, as well of their position in the process, in the two following events lists [Greenberg 1972]:

- Future Events List which includes those entities whose future time of re-activation is known (FEL);

- Current Events List which includes those entities whose re-activation time coincide with the current simulation clock time and those entities whose progress is delayed until certain conditions are met (CEL).

The executive task will consist of:

- find the entities with the minimum re-activation time in the Future Events List and move their records to the Current Events List;

- update the simulation time;

- scan the Current Events List and try to move the entities until they are blocked; if the entities are delayed because certain conditions are not met then they stay in the Current Events List; else if the entities are delayed for a fixed known time then they will be moved to the Future Events List; update if necessary the entities entry records.

The process interaction approach can be very attractive because, as it was said before, it is close to the intuitive approach of a new simulation user. On the other hand, the process implementation routines tend to be more complex and difficult to maintain than the event and activity routines. Also, the process interaction-based executive is very complex though this could not be a disadvantage as usually is hidden from the user in the simulation language or in the simulation subroutines library. Two of the most popular languages using this approach are SIMULA [Hills 1973] and GPSS [Greenberg 1972].

2.2.5. Three Phase Approach Modelling

One of the major drawbacks of the activity approach is its runtime inefficiency in what regards the unnecessary scanning of some activities. Nevertheless the activity approach has a very wide appeal due to its simplicity. Tocher [Tocher 1963] introduced the three phase approach which increases the efficiency of the activity approach and still keeps its simplicity. Tocher realized that there are two kinds of activities: 'B' activities which can be scheduled to a future simulation time and whose execution is controlled by the executive as it happens in the event-based approach; and 'C' activities which must be scanned and are executed only if certain conditions are satisfied as in the activity approach. This approach is called the three phase approach because its implementation has the following three distinguished phases:

- phase A which selects the next event and updates the simulation clock for the time that event is due;

- phase B which executes all 'B' activities scheduled for the current simulation time;

- phase C which scans the 'C' activities and execute them if the necessary conditions are met; this phase is repeated until no more activities can be executed.

The phases A and B are similar to the work done by the event-based executive and phase C is identical to the activity scanning of the activity-based executive. Figure 2.5 shows a logical diagram for a three phase-based executive.

Perhaps the best characteristic of the event approach is its runtime efficiency and of the activity approach its simplicity. The three phase aim to combine the strong aspects of the two: try to reach the runtime efficiency of the event approach without losing the simplicity of the activity approach. If the runtime efficiency is fundamental and the model complexity permits, the event approach is a good choice. If not, the three phase approach should be the best alternative.

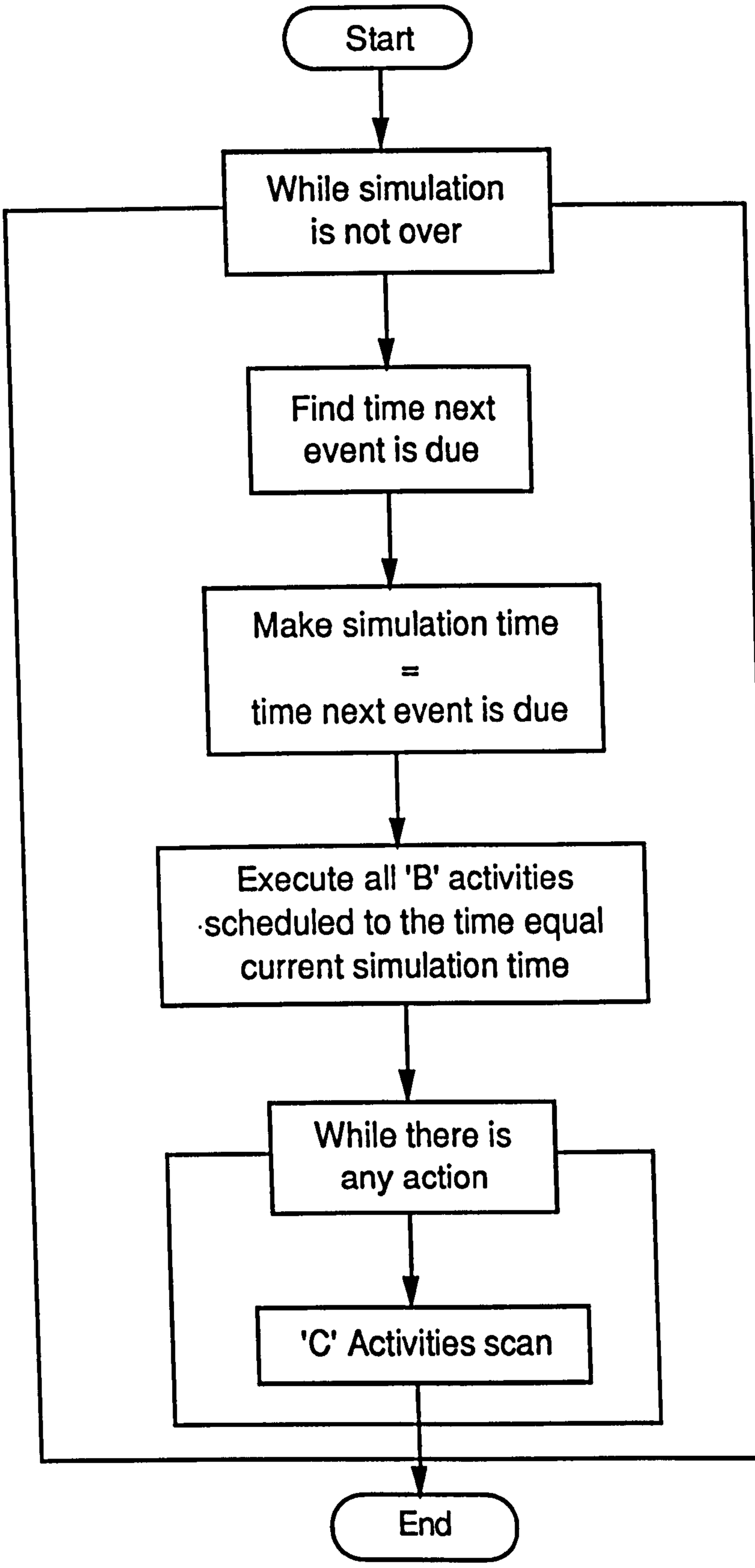


Figure 2.5 Logic diagram for a three phase-based executive

The process interaction approach could be adequate for a certain class of problems but generally it is quite complex and is difficult to deal with all the interactions between the different process. The activity approach, has been supplanted by the three phase approach which is now generally used [O'Keefe 1986]. Some of the packages that used this approach were FORSSIGHT/WITNESS [Hollocks 1983b] and SEE-WHY [Fiddy et al. 1981].

2.3. Visual Interactive Simulation (VIS)

2.3.1. History and Major Developments

A Visual Interactive Simulation (VIS) is a simulation which produces a dynamic display of the system model, and allows the user to interact with the running simulation [O'Keefe 1987]. The idea of using graphics, animation and interactive techniques in simulation has taken some time to become established. Even so, examples of the use of VIS techniques have been found since the 1960s.

O'Keefe [O'Keefe 1987b] has given as an example of the very early use of VIS the work of [Amiry 1965]. Hurrion [Hurrion 1986] has presented as describing early graphics facilities for simulation the work of [Donovan et al. 1968], [Harverty 1968], [Bell 1969] and [Katske 1975]. Hurrion has also mentioned the work of [Sulonen 1972] that has used graphics to build block GPSS simulation models and the work of [Sohnle et al. 1973] that has suggested general requirements for interactive simulation. Bell [Bell 1985] presents the work of [Alemparte et al.1975] and [Palme 1977] as examples of using moving displays driven by simulation models.

Although all these early examples the present approach to simulation known as VIS was conceived by Hurrion and it was described in his Ph.D. thesis [Hurrion 1976] and in a subsequent paper [Hurrion & Secker 1978]. Hurrion [Hurrion 1976] developed interactive simulation and animation methods for solving problems in the operational and production areas.

One of these problems was the job-shop scheduling in manufacturing [Hurrion 1978]. Hurrion found that improved solutions could be obtained by using a visual, dynamic representation of the job-shop problem when compared with the traditional batch

simulation. He has produced an extension to the simulation programming language SIMON [Hills 1965] specifically for VIS that was called VISION. This led to subsequent research and work from which results applications in the manufacturing industries [Brown 1978], process chemical plants [Fisher 1982] and automotive industry [Fiddy et al. 1981]. Most of this work was done in collaboration with major manufacturers such as I.C.I. and Rolls-Royce.

The early work done in Visual Interactive Simulation has demonstrated the advantages in allowing the user to make decisions during the simulation running. Although this has been important for the development of VIS its success and rapid growth was probably due more to the great improvement in the model-user communication achieved by its use. The visual and interactive aspects of VIS allowed the user to have a better understanding of the model and take an active part in using and experimenting with it. This led to an increase of confidence in the model and its results, and made easier the task of persuading managers to implement the solutions (one picture is worth a thousand words).

Another aspect where the use of VIS offered obvious advantages against batch simulation was validation. The simulation model was no longer regarded as a 'black-box' and the visual inspection of the model running has been revealed as an effective way of validating the model.

The developments in computer hardware and software have also contributed to the rapid expansion of VIS. The increase in computer power and the cost effective use of colours and graphics together with the generalized utilization of microcomputers were important for the expansion of VIS. Also important, was the developing of easy to use software and the improvements made in the software user interface.

One of the first commercially available (1979) VIS packages was SEE-WHY [Fiddy et al. 1981] developed at the Operational Research group at British Leyland. SEE-WHY was a set of FORTRAN subroutines that supported the development of VIS models. Originally SEE-WHY worked on a Cromenco Z-80 microcomputer with a I.S.C. Intecolor graphic display microcomputer. The availability of colours and semi-graphic facilities enabled the building of much more sophisticated simulation displays which had a great impact on the simulation users.

FORSSIGHT [Hollocks 1983a], released in 1981, was another VIS package developed by the OR group of the British Steel Corporation. This package had origin in the seminal work of Tocher [Tocher 1964] at the United Steel Companies. Tocher has worked, since 1957, in the development of computer simulation models for steel plants. The first simulation models were developed using machine code on the Ferranti Pegasus hardware. The concepts used in the steel plants simulation models were then extended and became the General Simulation Program (G.S.P.) [Tocher 1964]. G.S.P. was later implemented in a high level programming language to cope with the necessity of adapting G.S.P. to different hardware within British Steel Corporation.

In 1971, a FORTRAN version of G.S.P. III was produced which was known later as FORSS (FORtran-based Simulation System). FORSS has been enhanced taking advantage of the gained experience and evolving computer technology but always oriented toward printed-paper output. Although useful, printed output was rather limited in communication. This was recognized and some solutions were used to overcome the problem.

Hollocks [Hollocks 1983a] described the use of a schematic display board, representing a steel plant, that was manually updated according to the output of the simulation model. He also mentioned the use of physical displays where clocks, dials and lights were driven directly by the simulation model. Following the British Steel Corporation own experience and the results from the research work at Warwick University [Hurrion 1978] FORSS started to be enhanced, in 1980, to enable animated colour diagrams, interaction facilities, ease of use and even interactive means of constructing simulation displays. The new package was called FORSSIGHT and it was operational first on I.C.L. and I.B.M. mainframes.

With the developments in microcomputers it was clear, at British Steel Corporation [Hollocks 1981], that they had enough power to support the running of simulation models. As portability was becoming a desirable feature and recognising the microcomputers advantages it was decided to implement FORSSIGHT on this type of hardware. The I.S.C. Intecolor display unit was used connected with the Cromenco or Sage microcomputers the same type of hardware used by SEE-WHY [Fiddy et al. 1981].

FORSSIGHT although similar to SEE-WHY (both were VIS FORTRAN based packages) offered improved visual facilities. Furthermore, FORSSIGHT had additional features that made easier its use. FORGE [Hollocks 1983b] was a FORSSIGHT companion interactive package that allowed the user to build simulation models in a question-and-answer mode at the terminal. The FORSSIGHT development group established as Business Science Computing Limited and later on became part of Istel Limited.

Another package developed along the guide-lines of SEE-WHY was OPTIK [Insight International 1983] which was first released in 1983. OPTIK had origin, in 1982, when Bright, Elder and Fiddy left British Leyland and formed Insight International Limited. OPTIK shared some similarities with SEE-WHY as it was a FORTRAN based VIS package and used the same type of hardware including the I.S.C. Intecolor display terminal.

OPTIK was developed further and extended to become a more general purpose package for Visual Interactive Modelling (VIM). VIM [Bell 1985], [Hurrion 1986] is a generic term used for grouping the range of interactive and graphical model-building methods. OPTIK was conceived in a modular design with each module being dedicated to a particular task. The modules related to simulation were OPTIK-1, OPTIK-2 and OPTIK-11. OPTIK-1 consisted of a set of general interactive graphics FORTRAN routines. OPTIK-2 was a relational database that could be accessed using a query language or FORTRAN routines. OPTIK-11 enabled the building of visual interactive simulation models. Some problem-specific packages, which required no programming, were also developed such as Assembly Simulator, Process Line Simulator, Capacity Planning System, Job Shop Scheduling and Process Scheduling.

Crookes [Crookes 1982] at the Lancaster Centre in Simulation describes the work that has been done in VIS following the purchase of some APPLE II equipment. Based on this equipment, Herschdorfer developed a mixed continuous/discrete executive, Ellison worked on the visual display and Tunnicliffe-Wilson produced the simulation interactions [Ellison et al. 1982]. They have developed a simulation system that graphically showed the sensitivity of hospital waiting lists to the hospital system resources.

Following a demonstration of the work done showing the potential of VIS and microcomputers a collaboration was established with Leyland Vehicles Ltd. This led to the development of a VIS model of a conveyor system and the cranes servicing an automatic warehouse associated with a new assembly hall development at Leyland Vehicles Ltd [Crookes & Valentine 1982]. The advantages of having a pictorial representation was emphasized when for the first time the warehouse manager saw the working model [Crookes & Valentine 1982] "He was immediately able to discuss with us whether we had accurately represented the rules at various critical points and to satisfy himself, from the screen, that we had. There was neither a credibility gap nor a jargon wall".

The developments in computing and the advantages of VIS has made it one of the most used techniques for discrete simulation modelling in the past few years. Following packages like SEE-WHY [Fiddy et al. 1981] and FORSSIGHT [Hollocks 1983a] many others have incorporated graphics, interaction and animation features.

The Extended Control and Simulation Language (ECSL) [Clementson 1982] has been upgraded to provide some visual facilities. The Hand Or Computer Simulation (HOCUS) [Poole & Szymankiewicz 1977] evolved into a VIS package. SIMSCRIPT II.5 [Kiviat et al. 1987] included some extensions to support interactive graphics animation. The Extended Simulation System (TESS) [Standridge et al. 1987], to be used with the Simulation Language for Alternative Modelling (SLAM II) [O'Reilly & Lilegdon 1987], was developed to provide an environment assisting with model development, data management and graphical animation of simulation models. CINEMA [Healy 1986] was created as a graphical interface, microcomputer based, that allowed users to build highly detailed animation of any SIMAN (SIMulations ANalysis language) [Pedgen 1984] simulation model.

The emphasis in the development of simulation software has been upon ease of use and providing an integrated simulation environment. Shannon [Shannon 1987] identified the following objectives on which simulation developers have been focusing their attention: (1) reduced model development time; (2) improved accuracy; and (3) improved communication. Different approaches were taken aiming these objectives. Mathewson [Mathewson 1989] classified these approaches as Application-specific Systems, Simulation Pre-processors and Support Environments.

The Application-specific Systems have also been called Generalised Simulators [Mills 1988], Data-driven Generic Models [Pidd 1988] or just Generic Models [Crookes 1987]. Mills [Mills 1988] has defined a Generalised Simulator as "a validated model of a particular system framework, which the user tailors to his or her requirements by means of the input data". Most of the Application-specific Systems, specially in the manufacturing area, have made extensive use of interactive techniques, animation and graphics.

Examples of these were PC Model [SIMSOFT 1986], a general-purpose modelling system for the analysis of queuing networks, and WITNESS [Gilman & Watremez 1986], a manufacturing simulator that was developed from SEE-WHY. The level of generality of a Generic Model and its ease of use have been two factors that tend to conflict. An increase in the range of systems handled by a Generic Model, usually, makes it more difficult to use. Examples of Generic models restricted to specific applications areas are VISUALPLAN [Moreira da Silva & Bastos 1984] designed for simulation of Flexible Manufacturing Systems (FMS) and SIMAHID [Guimarães et al. 1985] for simulation of hydroelectric power dams.

The Simulation Pre-processors has been also called Code Generators [Mills 1988], Interactive Program Generators [Pidd 1988] (as program generators are usually interactive), or just Generators [Crookes 1987]. Mathewson [Mathewson 1989] defines the Program Generators as "interactive software tools that translate the logic of a model, described in a relatively general and simple symbolism, into the code of a simulation language or the appropriate list of building block definitions". Examples of Interactive Program Generators are CAPS [Clementson 1982] which produces ECSL code and DRAFT [Mathewson 1985] which can produce code in different languages.

The Support Environments were defined by Mathewson [Mathewson 1989] as "systems which aid all aspects of the use of simulation by providing individual modules for model building, control of experiments, results analysis and results presentation". TESS [Standridge et al. 1987] was developed as a support environment for SLAM II [O'Reilly & Lilegdon 1987], MAP/1 [Miner 1986] and GPSS/H [Crain et al. 1987]. TESS has general interactive capabilities and offered graphs and animation for the presentation of simulation results.

2.3.2. VIS Concepts

Over the past few years simulation developers realised the importance of incorporating animated graphic displays in their models. The use of animated graphics displays was became better known in the UK as Visual Interactive Simulation (VIS) [Hurrion 1976] and in the United States as animation.

Although, the VIS concept was fundamentally different from animation because it allowed user interaction with the model while the simulation was running. The animation approach just enabled the user to watch the progress of the simulation without allowing him to interact with the model running. The Interactive Simulation approach has been also called 'gaming' (or 'production gaming') [Hollocks 1983b] which highlighted the type of involvement required from the user. Hurrion [Hurrion 1989] has presented the technique of Visual Interactive Simulation as consisting of:

- developing a simulation model of the system under investigation;

- incorporating a method of animating the model using one or more colour graphics terminals. The dynamic animated view of the model ensures that the client commissioning the original study can observe the model in the form of a 'video' film. This has been shown to be of considerable help in removing the communications barrier that may exist between analyst and client with regard to the modelling technique and its assumptions. If a client can observe how a model is progressing through time he/she is in a position to:

- interact with the model in order to explore alternative decision strategies.

Hurrion [Hurrion 1989] has also identified the key components of VIS as having a visual dynamic representation of the model and the ability to interact with it.

One of the most important aspects of VIS was the fact that it has provided a way to improve the communication between the model and the user and also to get the user involved with the development and use of the model. The developments in hardware and software have represented an important role in this process.

The increasing use of computers, due to their ease of use and affordable prices, made them available, specially in the case of microcomputers, to people outside the Computer Centre. So, there was a necessity for adapting the software being developed to people not familiar with computers. This led to more user friendly operating systems and applications. The importance of the user interface has grown and a rapid evolution has been seen in this area. This had, of course, implications in the simulation systems. The simulation languages started to support interactive facilities, graphics and animation. Perhaps, the approach where the user interface had more importance was the application-specific systems that allowed the building of simulation models without or with minimum programming effort.

The quality of a VIS system has been much related to the way the running model was displayed on the screen and to the way the user interaction was handled. In the past few years the number of facilities provided by the VIS systems has been growing as well as their quality. Usually, a VIS system provided facilities for [O'Keefe 1987]:

(1) Visual Output - displaying the dynamic behaviour of the model and for presenting the simulation results;

(2) User Interface - allowing the user to interact with the model while running;

(3) Visual Input - creating a model visually.

As defined by Hurrion [Hurrion 1989] a true VIS system must incorporate at least (1) and (2). The consideration of only (1) in a simulation system is known as animation, an approach mainly supported in the United States. A system does not need to provide (3) to be considering a VIS system. Moreover, not many VIS systems have been providing facilities for Visual Input. Nevertheless, the importance of (3) in VIS systems has been growing specially within the application-specific systems (simulators) approach. This was due mainly to the general tendency of building easy to use software which led to an increasing importance of the user interface.

One area where this was particularly relevant was the manufacturing systems area. Many packages were developed specifically for simulating manufacturing systems and, because of that, they were able to make extensive use of Visual Input facilities to define

the simulation model. A good example of this was SIMFACTORY II.5 [CACI 1990]. With SIMFACTORY it was possible to build simulation model of factories or manufacturing systems using a menu-driven user interface and interactive facilities for creating icon-based displays.

. (1) Visual Output

The objective of Visual Output facilities provided by VIS systems are to enable the creation of a dynamic display that represents the behaviour of the running model. The display is created using graphic or semi-graphic facilities to build icons that form a 'mimic' diagram of the real system. The way the display is created depends on the type of solution chosen for building the simulation model.

If a programming language was used together with a simulation library like SEE-WHY [Fiddy et al. 1981] or OPTIK [Insight International 1983] then there were subroutines available to create the display and to do its animation. This approach had a great flexibility but involved, usually, a large development effort. To facilitate the task of creating the simulation displays special tools have been developed.

FORSSIGHT [Hollocks 1983a] has an interactive program to build graphic displays that could be saved and used later during the running of the simulation. Simulation languages have started to provide facilities to develop simulation models using graphics displays and animation. SIMGRAPHICS [CACI 1988b] was created as a language extension of SIMSCRIPT II.5 [Kiviat et al. 1987] for enabling the use of animated interactive graphics. SIMGRAPHICS includes a Graphics Editor that allows the user to draw shapes on the screen with a mouse and build forms to manage interactions.

The application-specific systems is an approach where the definition of the simulation output display can be done with less effort. This is due to the fact that these systems being specific for certain areas can provide special facilities for that purpose. The manufacturing area is a good example where packages like WITNESS [Gilman & Watremez 1986] and SIMFACTORY II.5 [CACI 1990] provide standard icons and menu-driven interactive facilities for building the

simulation displays and handling the movement of objects during the running of the simulation.

. (2) User Interaction

The objective of User Interaction is to allow the user to interact with the running model and change model parameters watching the effects without the need for restarting the simulation. As with Visual Output the way the simulation interactions were handled depended on the approach used for building the simulation model. The simulation subroutines libraries and the simulation languages provided facilities for allowing the analyst to develop the required interactions. The analyst has, with these approaches, all the flexibility to create the necessary interactions. The application-specific systems offered a set of standard interactions that usually covered the user needs. Although this approach has some advantages it can be limited, if the available user interactions are insufficient to handle the user needs.

. (3) Visual Input

The Visual Input although not mandatory in a VIS system had a great importance in improving the user interface of simulation systems. Since the development of the first simulation languages that work has been done in using graphics to facilitate the definition of simulation models.

One of the earliest attempt at graphical specification was GPSS [Greenberg 1972]. In GPSS a simple block diagram models the flow of entities through a network. Subsequently other languages had incorporated this same approach as SIMAN [Pedgen 1984] and SLAM [O'Reilly & Lilegdon 1987]. In HOCUS [Poole & Szymankiewicz 1977] activity cycle diagrams were used to formulate the simulation models. A HOCUS model consists of a block by block description of the activity cycle diagram. Whenever the hardware allows, an interactive graphical environment is used to build the model.

The block diagram is the most common approach adopted by general-purpose simulation languages with the objective of easing the task of programming the model. It was always difficult to balance the ease of use with the range of

applications covered by a simulation system. Usually the more general a simulation system, the more complex it becomes to use. As a consequence of this, the application-specific approach is the one where greater progress has been made with respect to specifying a model using Visual Input.

The manufacturing area due to the increasing complexity of the automated systems saw a rapid expansion in the use of simulation. The need to quickly obtain answers and to extend the use of simulation to non-specialists led to the development of specific simulation systems for the manufacturing area.

Systems such as WITNESS [Gilman & Watremez 1986] or SIMFACTORY II.5 [CACI 1990], using Visual Input, have reduced the time to build the models and required no specialized programming skills. In these systems the Visual Input technique used to describe the model is based on pre-defined icons which are placed and linked using interactive facilities. Whenever necessary, additional characteristics of the defined objects are entered in special data screens which are prompted to the user. Usually the display created during the Visual Input model specification is also used during the running of the simulation.

Visual Interactive Simulation (VIS) aggregates a set of graphics, visual and interactive techniques which can be incorporated to different levels of extent in a simulation system. In the past few years, there has been a large number of situations where VIS was successfully applied to different areas. Although much has been said about the advantages, problems and experiences of using VIS there was not much work done in terms of defining a methodology for its use and when it should be used. O'Keefe [O'Keefe 1987] has explained this fact by the existence of different methodologies for supporting decision making depending on the perspective of each VIS developer. O'Keefe [O'Keefe 1987] has identified at least the following methodologies:

Statistical or Traditional	- VIS is used as a selling aid and decision is still supported by statistical experimentation;
Decision Support	- VIS is used as a decision support method helping the user to interactively solve semi or ill-structured problems;

Computer Aided Design	- VIS is used to design a system by coupling together pre-defined parts;
Gaming	- VIS is used in gaming mode and the user benefits by learning, not by analysis of results;
Simulator	- VIS is used as a simple simulator.

O'Keefe [O'Keefe 1987] has given examples in the use of these different methodologies and concluded of the necessity for a single unifying methodology which should incorporate elements from the all the above methodologies. According to O'Keefe this will have the advantages of knowing when VIS is appropriate, increase its potential, form a base for the development of future packages and establish a common consensus on future research.

The advantages of the visual and interactive aspects in simulation models were recognized by the Operational Research community and have been successfully applied to other type of models like mathematical or heuristic models. This has given origin to a new modelling approach which was called Visual Interactive Modelling (VIM). In VIM a range of interactive, graphical model-building methods were used to build a computer based model of some problem situation. The VIM model has a visual dynamic representation and enable the user to interact with it in order to experiment alternative decision scenarios. One of the earliest examples of VIM use was the work of Hurrion [Hurrion 1980] in developing a visual interactive method of improving solutions for the travelling salesman problem. Bell [Bell 1985] and Hurrion [Hurrion 1986] have both given good reviews of VIM. They have highlighted its major aspects, have presented several examples of its use and have also analysed its perspectives for the future.

2.3.3. The Advantages and Pitfalls of Using VIS

In the past few years, the use of Visual Interactive Simulation (VIS) has been preferred over traditional batch simulation. The term batch simulation is used, as opposed to VIS, to name a simulation that does not make use of visual or interactive facilities while

running the model. The advantages of VIS have been largely published in the literature concerning simulation packages or the development of specific models.

Withers and Hurrion [Withers & Hurrion 1982] have written the following about the VIS benefits: "the tasks of validation and verification are eased, and (perhaps more importantly) the project's sponsor is encouraged to take an active part in the proceedings". Much of what has been said about VIS benefits has been related with the improved communication resulting from its use. This communication is established between the different elements involved in the simulation project model-analyst, model-user/manager and analyst-user/manager.

In the traditional batch simulation all the simulation work was done by the analyst, usually an Operational Research (OR) specialist, with no or with minimum intervention of the managers. This caused difficulties to the analyst to assure himself and to convince the managers that the model was a good representation of the real system. Also, when it was necessary to implement the solutions suggested by the analyst, based on the simulation results, again difficulties arose in convincing managers of their effectiveness.

During the model development stage the use of VIS eases the task of model validation and verification. The ability to display the model dynamically enabled the easy detection of logical errors. Since the manager could watch the model progress and interact with it he was able to evaluate and contribute to its correctness. The manager was, in this way, involved in the development process which increased his confidence in the model. During the experimentation stage the use of VIS allowed the manager to use is practical experience in order to interact with the model and try different strategies and alternative decision scenarios.

The impact of VIS on the managers and simulation end-users was well captured by several observations appearing in the initial work done by Hurrion or his students. Bell and O'Keefe [Bell & O'Keefe 1987] have presented the following ones:

- The picture has "a wide appeal" [Brown 1978];
- The picture gives the user "the freedom to shift attention" [Rubens 1979] between different parts of the simulation;
- "Situations may arise that ... the decision maker may never have envisaged" [Brown 1978];
- Users feel "participants rather than spectators" [Brown 1978];

- "If the model progresses as the manager expects, then credibility in its use is increased. If, however, the model diverges from the expectations of the manager then this leads to direct communications between the analyst and the manager" [Hurrion 1980].

All those observations reflect some of the benefits of VIS that were summarized by Bell and O'Keefe as follows: "beyond the value of VIS in allowing complex decisions to be deferred to the user, VIS was popular because it allowed users to understand the model and take an active part in using and experimenting with it" [Bell & O'Keefe 1987].

Crookes and Valentine [Crookes & Valentine 1982] have described the work carried out at Lancaster Centre in Simulation and Leyland Vehicles Ltd. and explained their experience of using interaction and animation for building simulation models on APPLE microcomputers. Crookes and Valentine have presented the following as some of the lessons learned from that work:

- "colour graphics, cheaply available and used carefully can overcome communication problems both between analyst and machine and between analyst and client";
- "confidence in computer simulation on the part of management, who often have much at risk, is not easily obtained, but the dynamic visual representation enables the none specialist to judge the correctness or otherwise of the modelling representation directly, leading to substantial involvement and easier implementation of the ultimate results" [Crookes & Valentine 1982].

Macintosh, Hawkins and Shepherd [Macintosh et. al. 1984] have given a good description concerning the development of visual interactive modelling philosophy in Ford of Europe. Macintosh et. al. have explained how the involvement of users in the process of building VIS models were fundamental to the success of the philosophy and have discussed their experience in the use of microcomputers for developing large VIS models. They have also highlighted the following benefits of VIS models:

- "Personalising the model - ... The user can modify the screen to his own personal taste and this allows him to make a contribution to the design of the model. Consequently, the user feels that the model belongs to him and greater enthusiasm is generated from the model";
- "Models as catalysts - ... a visual interactive model promotes new and original ideas, and can act as a catalyst for discussion between groups";

- "Model as training aids - Visual interactive models have been used to introduce line foremen to new facilities and working methods ... It is the industrial equivalent of a flight simulator" [Macintosh et. al. 1984].

Macintosh et. al. have concluded that VIS had opened up a new dimension in the communication not only between model developers and users but between users and managers.

Todd [Todd 1986] has described the experiences at British Steel (BSC) in the use of VIS as part of the decision making process in plant design and planning. Todd has summarized the following advantages offered by animated graphics facilities to the design engineer:

- "better communication of the simulation model's results";
- "greater understanding of the design problems encountered";
- "deeper involvement of management in the quantitative investigation";
- "a much more thorough examination of the proposed design results through the interest and ideas generated by this approach" [Todd 1986].

According to Todd [Todd 1986] VIS enabled BSC to evaluate ideas by viewing them in action before committing to them by investing capital or disrupting operations. Furthermore BSC's experience has showed that the use of VIS can not only significantly reduce the cost of plant development schemes but more importantly it can also benefit the reliability and performance of the final developments.

Bell [Bell 1989] based on a recent survey of visual interactive models-builders [Kirkpatrick & Bell 1989] has presented the following as the general opinion about the use of visual interactive models (mostly VIS models):

- "helps the manager or decision-maker understand the options, the system being modelled and the model itself";
- "enables the decision-maker to validate the model and achieve confidence in the decisions made using the model";
- "aids communication between modeller and decision-maker";
- "helps the model-builder understand the system or problem, and aids debugging the code";
- "leads to different (and better) decisions being made" [Bell 1989].

The observations presented here, and a lot more can be found in the literature, have highlighted the aspects where VIS has brought major benefits over traditional batch

simulation. According to O'Keefe [O'Keefe 1987] these benefits have resulted from the opportunity that VIS has given to developer and client to be involved in the following inter-related activities:

Selling - "VIS, and particularly visual output or animation, is a tremendous aid to selling the simulation method, a simulation model, or a specific solution";

Gaming - "Using model determined interaction, decision makers can be incorporated into the model, and decisions that are too difficult to encapsulate into a model can be referred to the user";

Learning - "In addition to being used as an analysis tool, a VIS can be used by a client to 'play' at managing a system. The benefit to the client is an increase in understanding system behaviour, and perhaps some information that can help solve an ill-structured problem, rather than an actual 'solution' to a 'problem'" [O'Keefe 1987].

As it has been demonstrated VIS has many advantages over traditional batch simulation. The power of interaction, colour graphics and animation have increased the demand for the use of these techniques in simulation projects. There were even situations where simulation projects were only accepted if they provided those facilities. However, VIS has some pitfalls that people should be aware of.

The use of quality dynamic displays has sometimes given a false sense of security about the correctness of the model which might cause an incomplete model logic validation. Also, the evaluation of a particular system cannot be made, in general, just by watching the simulation dynamic display for a short period of time. The use of VIS still requires the careful planning of the simulation experiments and a rigorous statistical analysis of the output results.

Another source of difficulties has been caused by the randomness in visual interactive simulation. Several authors have discussed the impact of different random-number seeds on the dynamic visual display of a VIS model. The most controversial aspect has been the fact that by simply repeating the run different results were shown on the screen due to the randomness effects. According to some authors this could destroy the manager's credibility on the simulation model.

However, Bell [Bell 1989] has mentioned that the use of traditional (i.e. non visual) stochastic simulation also required of the users an understanding of the effects of randomness. So, according to Bell, what was really necessary was a communication between analyst and manager in a way that the manager could understand some of the complexities of simulation studies. In this respect VIS was an excellent tool if used in the right way. Bell [Bell 1989] has also discussed the use of VIS to model transient system behaviour and proposed some approaches to stochastic VIS modelling to help overcome these difficulties.

CHAPTER 3

SIMULATION LANGUAGES AND THE APPROACHES TO MODEL BUILDING

3. Simulation Languages and the Approaches to Model Building

3.1. Introduction

A large range of simulation software is now available for a wide variety of computers. In 1986 the number of simulation software products was already over one hundred [Haider & Banks 1986]. With the increase of microcomputer power, specially of the IBM PC compatibles, the complexity of the simulation projects that they could handle has also increased. Most of the simulation packages were adapted to run on microcomputers and a large number was created specifically for them. The use of colours, graphics, animation and interactive techniques in simulation has become common during the eighties.

A good example of the success in the use of these techniques is the widespread use of the Visual Interactive Simulation (VIS) concept [Hurrion 1976] (see 2.3.1.). A growing number of simulation packages have incorporated the concepts used in VIS [Bell & O'Keefe 1987]. Animation is probably the technique which has more impact on the end users. The increasing graphics quality and computer power has made animation a key factor for the success of many simulation projects. The microcomputers, due to the availability of graphics for a reasonable cost, had an important role in all this process.

More recently, the graphic workstations appeared as a good support environment for simulation. Graphic workstations are becoming competitive because their price has been dropping and they offer high resolution colour graphics, Graphic User Interfaces (GUI) and an operating system, usually UNIX, which does not have the current limitations of the MS DOS operating system.

Manufacturing systems have been an area where extensive use of simulation has been made in the past few years. This has been caused by the greater complexity of these systems and by the development of more effective simulation software. Several software surveys have showed the large number of options available for using simulation to analyse manufacturing systems. A general overview was given by Haider and Banks [Haider & Banks 1986]. They have classified the simulation software at three levels: the system level (continuous or discrete); the application level (special purpose or general purpose); and the structural level (event, process interaction and activity scanning). Haider and Banks have also analysed the desirable features for

simulation software such as interactive debugging, materials handling features, animation, PC/mainframe compatibility and support environment.

The growing importance of animation and graphics in simulation has been demonstrated by the number of packages that incorporated these features. The factors to consider in choosing a graphically animated simulation system were identified by Grant and Weiner [Grant & Weiner 1986] as: model building (language, primary orientation and pre-processors); animation graphics; ease of constructing graphics screens; runtime interactions; and operational considerations (cost and hardware required). According to these factors, Grant and Weiner have also made a comparison between some of the commercially available packages in the United States.

The field of animation and graphics in simulation has registered a rapid evolution in the past few years, making any software survey very soon outdated. In a more recent work Law and Haider [Law & Haider 1989] have given some practical guidelines for selecting software for manufacturing systems as well as an updated software survey. Law and Haider have considered two major categories of software for simulating manufacturing systems. A general-purpose simulation language, where the model is developed by writing a program using the language modelling constructs. A manufacturing simulator for specific classes of manufacturing systems, where the model is developed with little or no programming.

Law and Haider have made an extensive vendor survey of 23 simulation products where the vendors were asked to comment on a large number of features. These features were grouped under the following headings: general features; animation; statistical capabilities; material handling modules; customer support; and output reports. The results of the survey were presented in two sets of tables, one set for the simulation languages and the other set for manufacturing simulators.

3.2. Simulation Systems

When a simulation model has to be built several options are available for that purpose. One of the first decision factors considered in the choice of a solution is, usually, the expertise of the persons who are going to be involved in the development of the simulation model. If the persons were simulation experts or end-users, wanting to

obtain a quick answer for a problem, the options to consider could be completely different.

The decision is often a result of the trade-offs between the ease of use and the complexity of the systems that a simulation package could handle. Normally, an application-specific simulation system tends to be easier to use and so more adequate to users not familiar with simulation modelling. However, its low level of generality or embedded assumptions can sometimes make it inappropriate to solve a particular problem.

In recent years, a large number of simulation systems have been developed. These systems can be grouped under the following categories according to their model building approach:

simulation libraries	- the model is built in a general-purpose programming language, like FORTRAN, using a collection of routines from a simulation development support library;
simulation languages	- the model is built in a language, similar to a programming language, but whose structure and commands are oriented to the development of simulation models;
graphical block languages	- the model is built by defining a flow diagram using pre-defined blocks;
program generators	- the model is built by specifying a simple description of it which is then used to generate the complete model code;
data-driven generic models	- the model is built by defining a set of parameters that characterised it;
support environments	- the model is built within an environment which aids the user in all aspects of the simulation project.

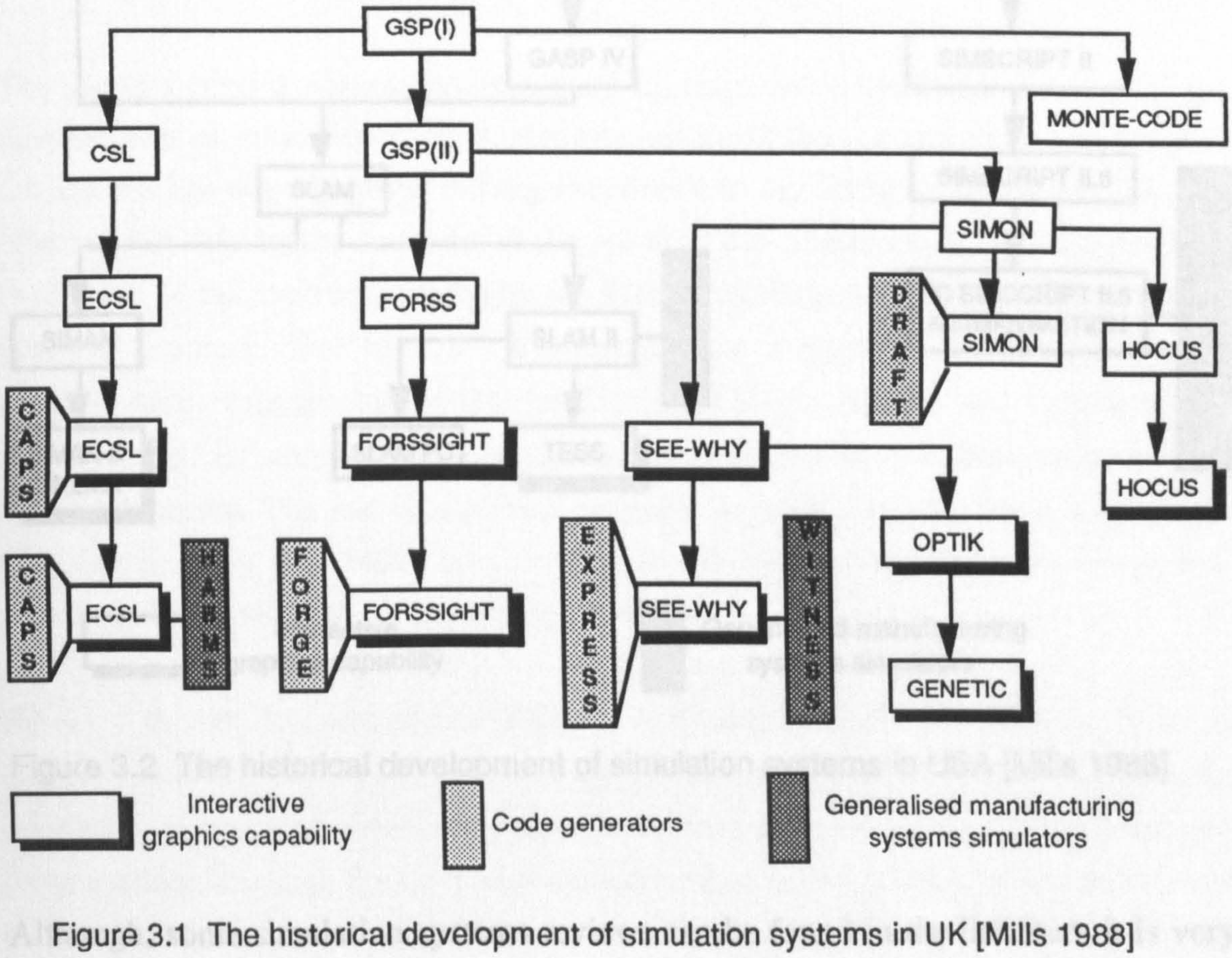


Figure 3.1 The historical development of simulation systems in UK [Mills 1988]

Most of the current simulation systems had origin in one of the two families [Mills 1988] - the material-based family, widely used in the USA, and the machine-based family used in the UK. The UK family uses the Three Phase approach (see 2.2.5.) based upon the concepts developed by Tocher [Tocher 1963] at the Operational Research Group of the United Steel Companies. The first language using this approach was the General Simulation Program (GSP) which was the base for the development of many others simulation systems.

In figure 3.1 a diagram is presented showing the progressive development of the Three Phase languages according to Mills [Mills 1988]. The US family development was based on the Q-GERT/GASP languages and a diagram showing its evolution, according to Mills [Mills 1988], is presented in figure 3.2.

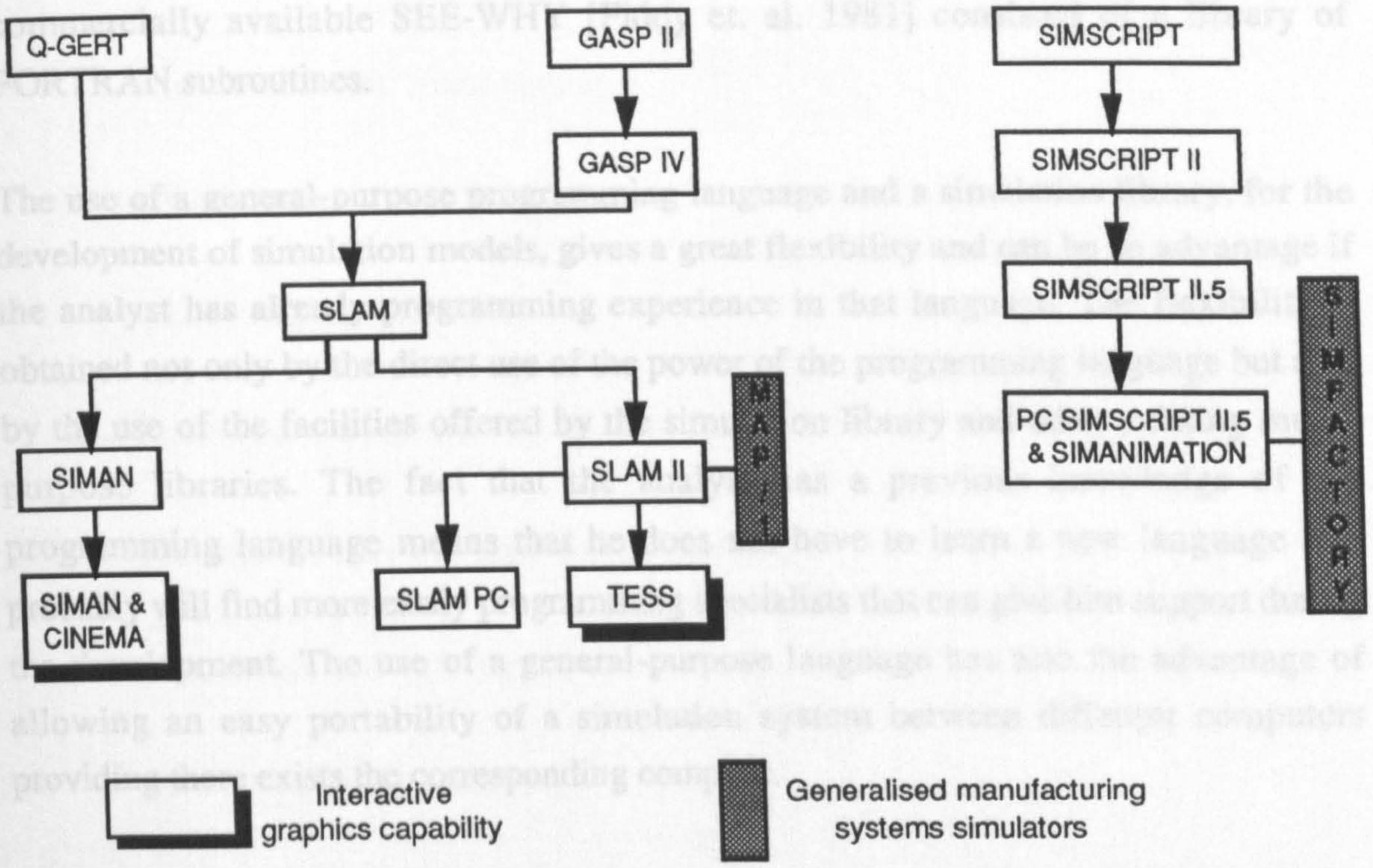


Figure 3.2 The historical development of simulation systems in USA [Mills 1988]

Although, some simulation systems reviews can be found in the literature it is very difficult to find a complete and up to date survey. This is explained by the large number of existing simulation systems and by the rapid evolution verified in this area. Next, the different approaches to model building will be described and examples of simulation systems using them will be given.

3.2.1. Simulation Libraries

Most of the early simulation models were developed using general-purpose programming languages. Soon it was realised that certain features were required by different models. The programming languages ability of enabling the organization of the program logic in different modules, named subroutines or procedures, which could be called repeatedly, made it possible to build a set of simulation support routines that could then be used in other simulation projects. The ALGOL procedures of SIMON [Hills 1965] and the FORTRAN subroutines of GASP [Pritsker 1974] are two well known examples of this approach. Also, the first Visual Interactive Simulation system

commercially available SEE-WHY [Fiddy et. al. 1981] consisted of a library of FORTRAN subroutines.

The use of a general-purpose programming language and a simulation library, for the development of simulation models, gives a great flexibility and can be an advantage if the analyst has already programming experience in that language. The flexibility is obtained not only by the direct use of the power of the programming language but also by the use of the facilities offered by the simulation library and other existing multi-purpose libraries. The fact that the analyst has a previous knowledge of the programming language means that he does not have to learn a new language and probably will find more easily programming specialists that can give him support during the development. The use of a general-purpose language has also the advantage of allowing an easy portability of a simulation system between different computers providing there exists the corresponding compiler.

However, the development of a simulation model using a general-purpose programming language requires, usually, a reasonable level of programming skills without which the long learning period can make the project become unpractical. Also, a general-purpose programming language does not have embedded simulation concepts that can help the analyst in developing faster and better structured simulation models. Furthermore, the absence of error messages and debug facilities oriented towards the simulation can make things more difficult during the development stage.

3.2.1.1. GASP IV

The program structure for discrete modelling has at the first level (see 2.2.1.) a control module called the executive. The executive controls the model execution flow by calling the subroutines in the sequence defined by the model logic. In GASP IV [Pritsker 1974] the tasks of the executive, such as determining the next event and advancing the simulation clock, are performed by the subroutine GASP. The user must supply a main program, an optional initialization subroutine INTLC, a subroutine describing the different events EVNTS and an optional report subroutine OTPUT. The subroutine EVNTS controls the calling of each event by the use of a computed GO TO statement. As GASP IV most other simulation subroutines libraries provided facilities such as an executive and a set of simulation development support tools. The GASP IV library includes routines to implement the following tasks:

- time advance and status update;
- initialization;
- data storage and retrieval;
- location of state conditions and entities;
- data collection, computation and reporting;
- monitoring and error reporting;
- random deviate generation;
- various miscellaneous routines.

This gives an idea of the type of functions that are performed normally by a simulation library. In appendix III.A a subroutine for the end of operation event of a simple job shop problem [Carrie 1988] is presented exemplifying the use of GASP IV FORTRAN subroutines. A more detail description of the GASP IV functions can be found in [Pritsker 1974] and a brief explanation in [Pidd 1988].

3.2.1.2. SEE-WHY

SEE-WHY is a FORTRAN based simulation subroutines library that allows the development of general purpose discrete event simulation models. SEE-WHY uses the Visual Interactive Simulation (VIS) approach offering special features for graphical display and runtime interaction. It was designed with the intention of overcoming communication problems in a simulation project between analysts and managers.

SEE-WHY models are written using the Next Event time handling technique (see 2.1.) and the Three Phase approach (see 2.2.5.). The main components of a SEE-WHY simulation model are a list of future events (the Timeset), a simulation clock, modelling elements (entities and sets) and routines for manipulating these elements according to

the model logic. The basic elements provided by SEE-WHY for modelling, statistics collection and probability distributions are the following [Istel 1987]:

entities - used to represent items which move from one discrete state to another as events occur during the simulation progress (e.g. pallets, transporters etc.);

sets - used to represent an ordered collection of elements (e.g. a queue of pallets to be picked, a conveyer etc.);

vessels - used to represent an element that act as a 'container' into and out of which a fluid flows; they can be used, under certain conditions, in modelling continuous processes;

timeseries - used as a collection of data which is recorded at predefined time intervals during the running of simulation and that can be automatically displayed (e.g. snapshots graphs of simulation progress);

histograms - used as a collection of data recorded during the running of a simulation and can be automatically displayed (e.g. cumulative totals to evaluate model performance);

distributions - used as a set of data from which SEE-WHY can take samples (e.g. system distributions, i.e. normal, exponential, etc., or user defined distributions).

When building a SEE-WHY model the analyst must first look to the real system and define the necessary simulation elements. Each simulation element must be described by its name, type, model number and detailed characteristics. If applicable, the analyst must also decide about its representation on the screen. Normally, a SEE-WHY simulation consists of the following steps [Istel 1987]:

- initialize displays;

- initialize main simulation data array;

- define simulation elements;
- initialize simulation state;
- schedule initial event;
- loop to advance time to next event, process it, and schedule subsequent events.

SEE-WHY provides subroutines to support the development of the functions involved with these steps. SEE-WHY handles the storage and management of simulation data, controls the advance of the simulation time and the scheduling and execution of events, provides standard displays and user-defined displays, and also has facilities to interrupt the running of a simulation and interactively inspect and modify simulation data. The user is responsible for defining the events (attempts to change the simulation state) in the simulation and establish the sequence which reflects the logic of the real system. The following two types of events are considered in SEE-WHY:

scheduled event - its an event that is scheduled and occurs after a defined period of time;

consequential event - its an event which occurs only if certain conditions, depending on the simulation state, are met.

The 'schedule events' correspond to the 'B' activities and the 'consequential events' to the 'C' activities of the Three Phase approach (see 2.2.5.). In appendix III.B a customer arrival event subroutine is presented [Istel 1987] describing the use of some of the SEE-WHY subroutines. A detailed description of the SEE-WHY subroutines can be found in [Istel 1987] and a brief overview in [Gilman & Watremez 1986].

3.2.1.3. SIMVIS

SIMVIS [Bastos & Moreira da Silva 1985] is a FORTRAN 77 event-based Visual Interactive general purpose system developed at the University of O'Porto. SIMVIS was used in the development of several Data-Driven Generic Models such as job-shop planning [Bastos 1985], Flexible Manufacturing Systems (FMS) [Moreira da Silva & Bastos 1984] and hydroelectric power dams [Guimarães et al. 1985] as well as specific

simulation models. In appendix III.C an event subroutine is presented from a SIMVIS based simulation model of a full automated warehouse. In chapter 5 the SIMVIS concepts will be explained in more detail and a description of its libraries will be given (see 5.3.1.).

3.2.2. Simulation Languages

A simulation language is a programming language whose problem orientation is towards the building of simulation models. The objective for the creation of a simulation language was to obtain a higher level programming language whose structure and instructions permit the analyst to concentrate on the logic of the system to be modelled, not on computer programming as such. Some of the first simulation languages were the General Simulation Program (GSP) developed by Tocher [Tocher 1963] at the Operational Research Group of the United Steel Companies, the Control and Simulation Language (CSL) [Buxton & Laski 1963] developed at Esso and SIMSCRIPT [Markowitz et al. 1963] developed at the RAND Corporation in USA.

The simulation library approach offers more power and flexibility then the other approaches but usually require more programming skills, simulation expertise and great development effort. The use of a simulation language has most of the benefits of the simulation libraries approach except in portability. The development of a model in a simulation language requires the existence of the corresponding compiler or interpreter for the specific machine and can cause some difficulties when porting the model to a different machine.

One of the major advantages of a simulation language consists in the use of concepts and instructions that are adequate to the building of simulation models. The simulation language instructions are usually more compact, due to its high level, more readable, often English-like statements and the simulation elements can be referred by name. This allows the development of quicker and better structured models requiring however, a high level of expertise from the analyst. The existence of error messages and debug facilities related directly with the simulation can also be of great help during the development stage. The need for long training periods or alternatively the need for a simulation specialist can be thought as the main disadvantage of this approach.

3.2.2.1. The Extended Control and Simulation Language (ECSL)

ECSL [Clementson 1982] was developed as an extension of the Control and Simulation Language (CSL) [Buxton & Laski 1963]. ECSL is an interpreted language that uses the activity approach (see 2.2.3.) for model building. The ECSL system is written in FORTRAN which facilitate its availability in different computers providing they have a FORTRAN compiler and a suitable configuration. An ECSL program is structured in the following sections:

definitions - the section where is defined the name of each class of entity, the number of entities in each class, the sets (queues) of which each class may be members and any variables, arrays, histograms or functions required;

initialization - the section where the initial state of the system is defined by specifying details of any activities in progress and the attributes of the entities involved;

activities - this is the main section of the program which consists of a sub-section for continuous process, a sub-section for recording the values of simulation parameters and a sub-section for describing each activity;

finalization - this section is called at the end of the simulation run and is used to perform any necessary calculations and print out the final report;

data - the section where initial values are assigned to variables and where it is performed the system initialization (e.g. queues contents, random number streams etc.).

In Appendix III.D a small ECSL program [Pidd 1988] is presented. The model represents a warehouse with three loading bays, each of which can accommodate one truck or two vans. A complete description is given in [Pidd 1988]. The print out of the ESCL model, in appendix III.D, is used to exemplify all the different sections described above. This example shows also some of the English-like ECSL instructions and how the simulation elements are referred by name. So, the statement 'THERE ARE 8

TRUCK SET WAIT OS' in the 'DEFINITIONS' section means that there are 8 entities of the class (type) 'TRUCK' and they may be in one of the sets (queues) 'WAIT' or 'OS'. The execution logic of an ECSL program is presented in the diagram from figure 3.3.

The diagram in figure 3.3 shows the order in which the different sections in an ECSL program are processed. A simple version of the work carried out by the ECSL executive is also presented. A more detailed description of an activity approach executive is given in 2.2.3..

In the 'ACTIVITIES' section all the model activities are described (see Appendix III.D). Each activity description block starts with the 'BEGIN' statement followed by the condition statements (test head). These condition statements, depending on the simulation state, decide if the activity actions will be executed or if the control passes to the next activity. The activity actions include computing the activity duration and the value of other variables, setting entities attributes, changing the state of entities (e.g. moving entities from one queue to another) and recording statistics. More information and examples of the use of ECSL can be found in [Clementson 1986], [Pidd 1988] or [Carrie 1988].

3.2.2.2. SIMSCRIPT

SIMSCRIPT is a English-like general-purpose simulation programming language. It was developed at the RAND Corporation in the early 1960s by Harry Markowitz [Markowitz et al. 1963]. Initially, it was conceived as an extension of FORTRAN to facilitate the development of simulation models. The simulation statements were translated to FORTRAN by the SIMSCRIPT pre-processor and then compiled as a normal FORTRAN program.

Later at Consolidated Analysis Centers, Inc. (CACI), Markowitz have done a major revision of the language, known as SIMSCRIPT I.5. This version was independent of the FORTRAN language and its statements were compiled directly to assembly language code. SIMSCRIPT I.5 was implemented in different computers and was used, for almost a decade, as a standard for building simulation models. An introduction to simulation modelling and a description, illustrated with several examples, of the

SIMSCRIPT language in the beginning of the 1970s can be found in Wyman [Wyman 1970].

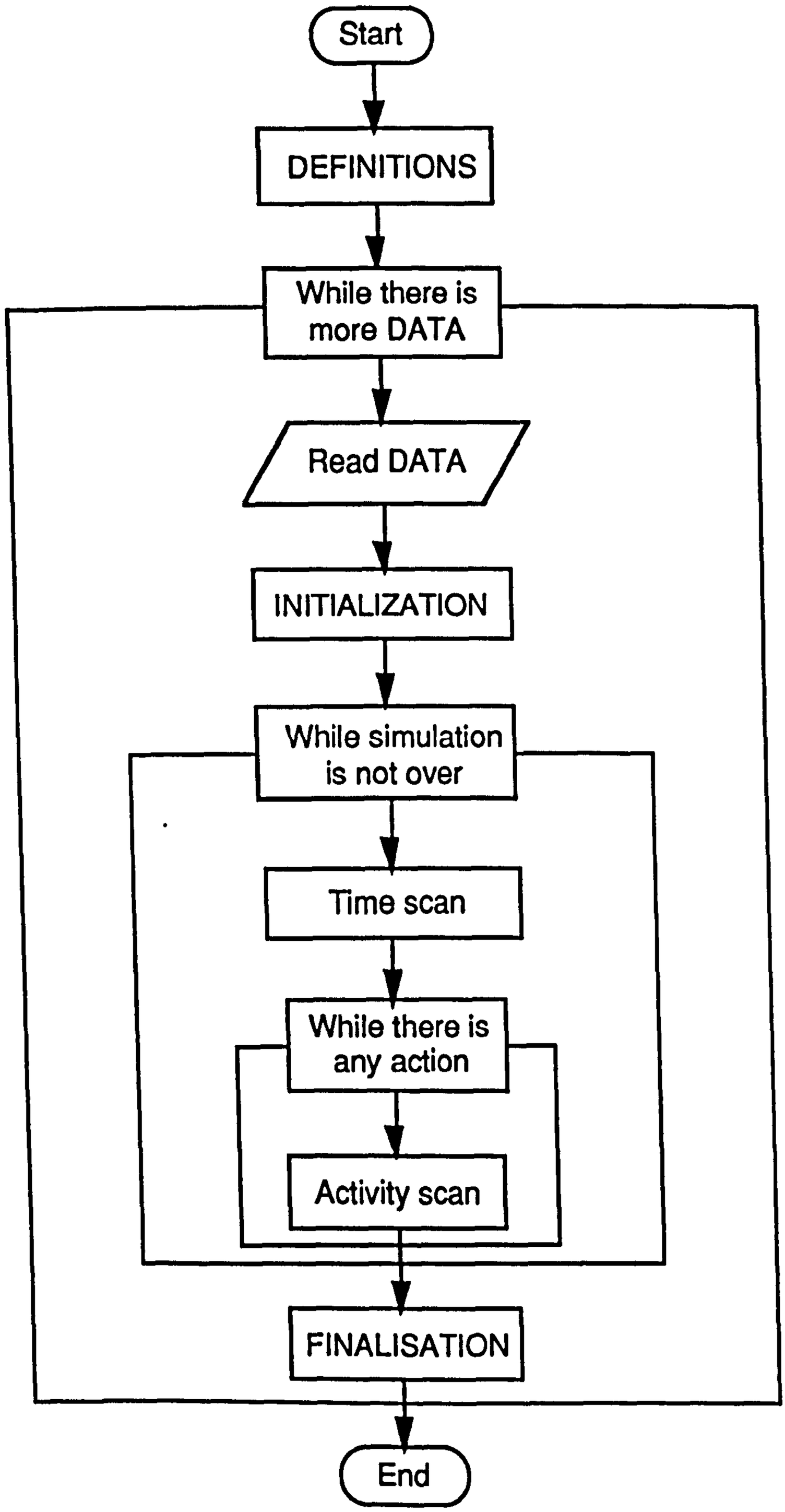


Figure 3.3 Logic diagram for the execution of an ECSL program

Based on the experience gained with SIMSCRIPT I Markowitz has designed a substantially advanced language named SIMSCRIPT II. Its first implementation was done at RAND. An improved version called SIMSCRIPT II Plus was created by a company known as Simulation Associates. This version was later acquired by CACI which produced SIMSCRIPT II.5 [Kiviat et al. 1987] the most recent version of this language. SIMSCRIPT II.5 is widely used today on a large variety of computers.

SIMSCRIPT was originally an event-oriented (see 2.2.2.) simulation language. In the middle 1970s CACI has introduced two new concepts, to SIMSCRIPT II.5, processes and resources. This has enabled the use of SIMSCRIPT II.5 as a process-oriented as well as an event-oriented simulation language.

A process represents an object and the sequences of actions it experiences during its existence in a simulation. The process is the basic concept behind the process interaction approach to simulation modelling (see 2.2.4.). Following, the main SIMSCRIPT concepts are described using the process-interaction approach.

A resource is a passive element of a model which is required by the process objects. If a resource is not available when required, the process object must wait, in a queue or waiting line, until the resource becomes available. This happens when the resource is released by the process holding it. The released resource is given to the first process object waiting for it, according to some priority rule, or simply made available if not required.

A SIMSCRIPT II.5 program structure consists of the following three sections:

preamble	-	the section where a static description of each modelling element is given;
main	-	the section where the program execution begins;
process routines	-	the section where each process routine declared in the preamble is described.

. Preamble

The preamble is the first section of every SIMSCRIPT II.5 model. It is used to declare or define characteristics of the elements used in the simulation program. All the processes, resources, entities, events and sets must be named in this section and also,

all global variables and statistics to be collected must be defined. The preamble section begins with the keyword PREAMBLE and ends with the statement END.

. Main

The execution of a SIMSCRIPT program starts with the first instruction in this section. The usual segments in the Main program are: the creation and initialization of the resources; the activation of the initial processes; the start of the simulation by the use of the START SIMULATION statement; and the definition of the statements to be executed after the simulation end (final output reports, new simulation run etc.). When the START SIMULATION statement is executed the control passes to the timing routine.

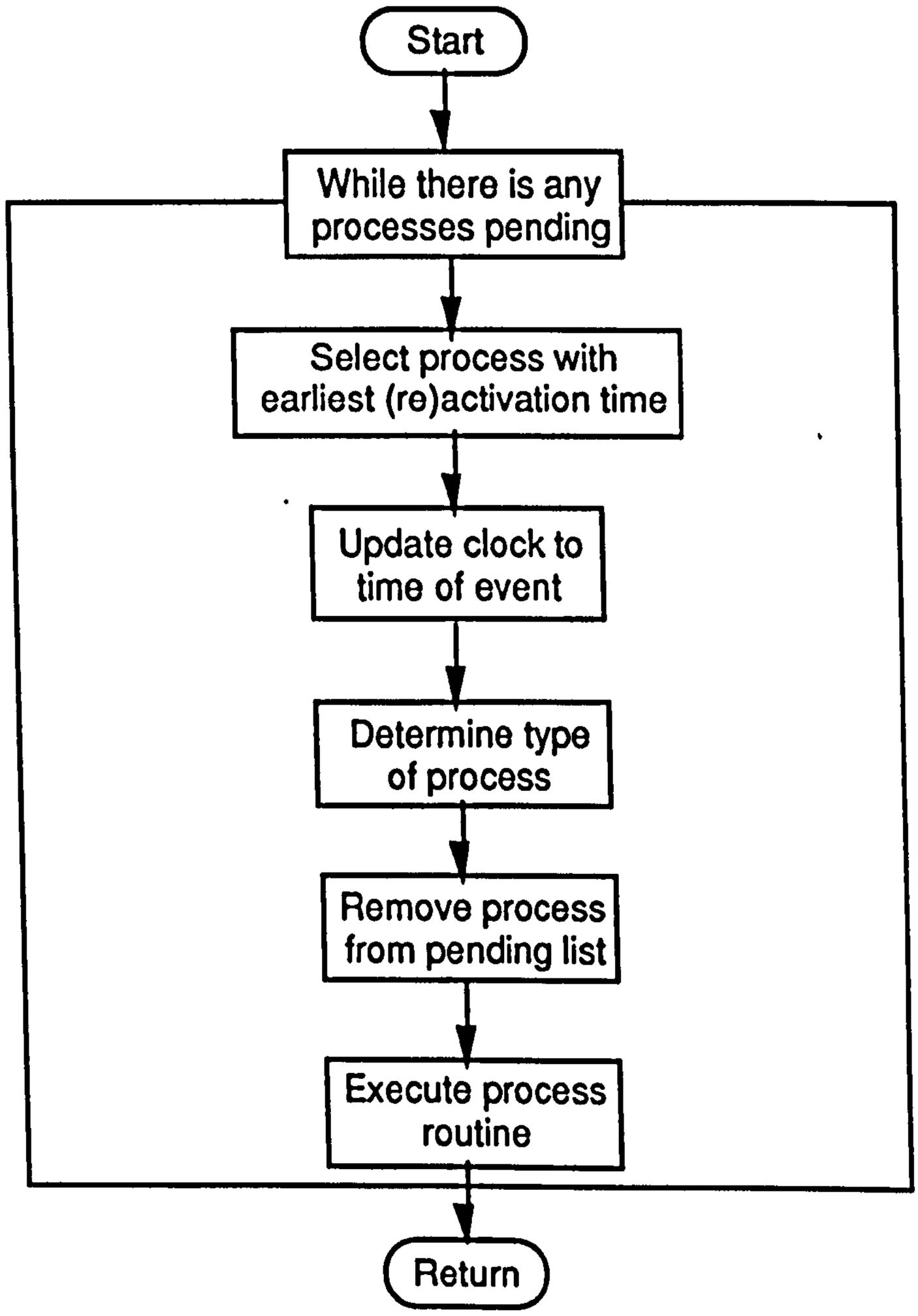


Figure 3.4 Logic diagram for SIMSCRIPT II.5 timing routine [Russell 1989]

The timing routine follows the logic of the process interaction-based executive (see 2.2.4.). The diagram from figure 3.4 describes the logic of the basic SIMSCRIPT II.5 timing routine [Russell 1989]. Any statements following the START SIMULATION statement will be executed only after the control returns from the timing routine. This happens when the simulation reaches an end either by running out of things to do or by the execution of an explicit stop statement in one of the processes.

For the simulation to start it is necessary that in the pending list there is at least one process (see figure 3.4). A process to be in the pending list at the beginning of the simulation must be activated in the Main section of the program before the START SIMULATION statement. The execution of the initial process or processes will normally activate other processes which execution sequence will constitute the simulation run. In the diagram of figure 3.4 the word 'event' is used as a pending (re)activation of a process.

. Process routines

In this section each process declared in the preamble must be fully described in a process routine. The process routine gives a complete description of the logic of the process. This includes the values of local variables, reactivation time, reactivation point, the resources the process currently holds and the actions to be executed when events occur.

Originally, SIMSCRIPT has used only the event-oriented approach to simulation modelling. Later, it has been improved to allow the building of models using the process-oriented approach. Currently, SIMSCRIPT II.5 supports the development of simulation models using either the event orientation, the process orientation, or both simultaneously. The major concepts involved in a SIMSCRIPT II.5 program were already introduced using the process-oriented approach and are still applicable when using the event-oriented approach. Nevertheless, as described in 2.2., there are some differences when using the event or the process as the basic building block of a simulation model.

As explained in 2.1 an event corresponds to the instant of time in which a simulation system change its state. The event is the basic element of a discrete simulation system in a way that the simulation progress from one event to the next. The process concept is

more elaborated and can be thought of as a sequence of events. In SIMSCRIPT II.5 an event is described by the following information [Russell 1989]:

event 'notice' - the data block containing relevant information about a particular copy of (or instance of) the event;

event routine - the program code describing the logic of the event, how it responds to its environment, and how it changes the environment;

global variable - a global variable, having the same name as the event, used to identify a particular instance of the event, either at creation time or during execution.

A major difference between the event and process concepts is that while the event is instantaneous, in relation to the simulation time, the process is not. As a consequence, the attributes and local variables of a process are preserved during its life while in the case of events local variables are initialized to zero, each time the event occurs, and the event notice self-destructs on entry to the event routine. If it is necessary to pass information between events a temporary entity needs to be used. SIMSCRIPT II.5 has two type of entities, permanent and temporary. Permanent entities stay in the system until the simulation ends. Temporary entities can be created and destroyed while the simulation is running.

Figure 3.5 shows SIMSCRIPT II.5 routines representing a telephone call using the process and event approaches [Russell 1989]. As can be seen by this simple example any time delay must be simulated, in the event approach, using at least two events. This makes modelling more complex using the event approach when compared with the process approach. Nevertheless, the use of the event approach results, usually, in the building of more efficient models (see 2.2.).

SIMSCRIPT II.5 allowing the use of process, events or both simultaneously in a model gives to the user the choice of the more adequate approach to his problem. In appendix III.E two versions of a SIMSCRIPT II.5 model of a petrol station are presented using the process and event approaches [Russell 1989]. The most important differences between the two implementations are [Russell 1989]:

- the generation of new ARRIVAL events, equivalent to the process GENERATOR, is made in the ARRIVAL event itself, although it could be made in a separate event;

- the CUSTOMER entity created in ARRIVAL is disposed of either by filing in the queue or passing to END.SERVICE (otherwise it would be lost to the system when the next call to ARRIVAL was done);

- the event END.SERVICE must have an attribute (ES.CUSTOMER) to store the pointer to the CUSTOMER with which it is associated in each call;

- in the event model more variables are required (i.e. NO.BUSY and NO.CUSTOMERS) than in the equivalent process model.

PROCESS APPROACH

```
PROCESS CALL
  ADD 1 TO BUSY.LINES
  WORK 5 MINUTES
  SUBTRACT 1 FROM BUSY.LINES
END
```

EVENT APPROACH

```
EVENT BEGIN.CALL
  ADD 1 TO BUSY.LINES
  SCHEDULE AN END.CALL IN 5 MINUTES
END
```

```
EVENT END.CALL
  SUBTRACT 1 FROM BUSY.LINES
END
```

Figure 3.5 SIMSCRIPT II.5 different approaches to a telephone call model [Russell 1989]

Comparing the two approaches and looking to the programs in appendix III.E it is clear that the use of the process approach results, usually, in more structured and easy to understand simulation models. A more detail description of the petrol station model and a thorough explanation of the fundamentals of simulation methodology through the use of SIMSCRIPT II.5 case studies can be found in [Russell 1989].

The SIMSCRIPT II.5 programming language concepts are fully described in [Kiviat et al. 1987] and a description of each of the language elements in alphabetical order is given in [CACI 1989]. Versions of SIMSCRIPT II.5 are available for most popular computers. The IBM PC and compatibles SIMSCRIPT II.5 version offers an interactive

environment, known as SIMLAB [CACI 1988], which controls all aspects of a simulation project. This version have also graphics and interactive facilities through the use of SIMGRAPHICS [CACI 1988b].

SIMSCRIPT is a very popular language and many authors have given brief descriptions and simple examples of its use. A small SIMSCRIPT II.5 tutorial was given in [Russell 1986] and a more complete one involving an animated graphic model of a mining operation which uses a lift to carry ore from the underground levels to the surface was given in [Russell 1987].

3.2.3. Graphical Block Diagram Languages

The use of a collection of subroutines, like GASP IV or SEE-WHY, or the use of a statement simulation language, like ECSL or SIMSCRIPT, requires that the analyst be a reasonably proficient programmer. Since the beginning of the development of simulation systems that a great effort has been made in the search for solutions to facilitate the analyst task. This work aimed to help the analyst in the development of simulation models and in the interpretation of its results. The way the simulation model is defined has been one of the main concerns of simulation systems developers. The objective has been to obtain simulation systems which allow, as much as possible, a quick and easy definition of the simulation model requiring minimum or no programming expertise.

The use of block diagrams was one of the early attempts to ease the analyst's programming task. With a block diagram language the analyst used pre-defined blocks to build a flow diagram describing the logic of the model. The block diagram approach was intended for use by non-computer specialists. Although, easier to use this approach to simulation is, usually, less flexible and can be unsuitable for certain types of applications. The first simulation system using this approach was the General Purpose Simulation System GPSS [Gordon 1961].

3.2.3.1. General Purpose Simulation System (GPSS)

The General Purpose Simulation System language GPSS has been developed over many years and is one of the most popular languages. The original version was presented in 1961 [Gordon 1961] with the name General Purpose System Simulator. Its

development has been supported, since the beginning, by IBM Corporation. GPSS has been implemented on different machines and has evolved through several versions.

In 1967, GPSS/360 was introduced and the GPSS name was changed to General Purpose Simulation System. GPSS V, a super set of GPSS/360, was released in 1970. This version contained several new features such as free form coding and an interface with FORTRAN and PL/1 programming languages. Both GPSS/360 and GPSS V were the result of successive versions developed by IBM.

Paralleling the IBM releases, GPSS has been implemented by other vendors in IBM and non-IBM hardware. Schriber [Schriber 1987] has given a list of the different GPSS implementations and a complete information about their vendors. Examples of those implementation are GPSS/H and GPSS/PC. GPSS/H, from Wolverine Software, is an upwardly compatible superset of GPSS V offering more advanced features [Crain et al. 1987]. GPSS/PC, from Minuteman Software, is an IBM PC implementation of GPSS offering an interactive environment supporting graphics and animation [Minuteman Software 1986]. A description of GPSS/PC can be found in [Cox 1987a], [Cox 1987b] and [Garzia 1986].

GPSS uses the process interaction approach (see 2.2.4.) for building the simulation model. Processes describe the movement of the temporary entities, called transactions, through the system being simulated. In GPSS, a model is described as a block diagram using specific block types (48), representing activities, and lines joining the blocks indicating the sequence in which the activities can be executed. Figure 3.6 shows some of the most used GPSS block and table 3.1 list the corresponding statements and parameters [Gordon 1978].

Transactions are created at one or more GENERATE blocks during the execution of a GPSS program. They move from block to block representing the sequence of events in the system and are removed at TERMINATE blocks. The parameters in the GENERATE A,B,C,D,E,F statement (see table 3.1) have the following meaning: A and B defines the range (A-B,A+B) of the uniformly distributed interarrival time; C is the time of the first arrival; D is the number of transactions to be generated; E is the selection priority; and F is the number of attributes (parameters) of each transaction.

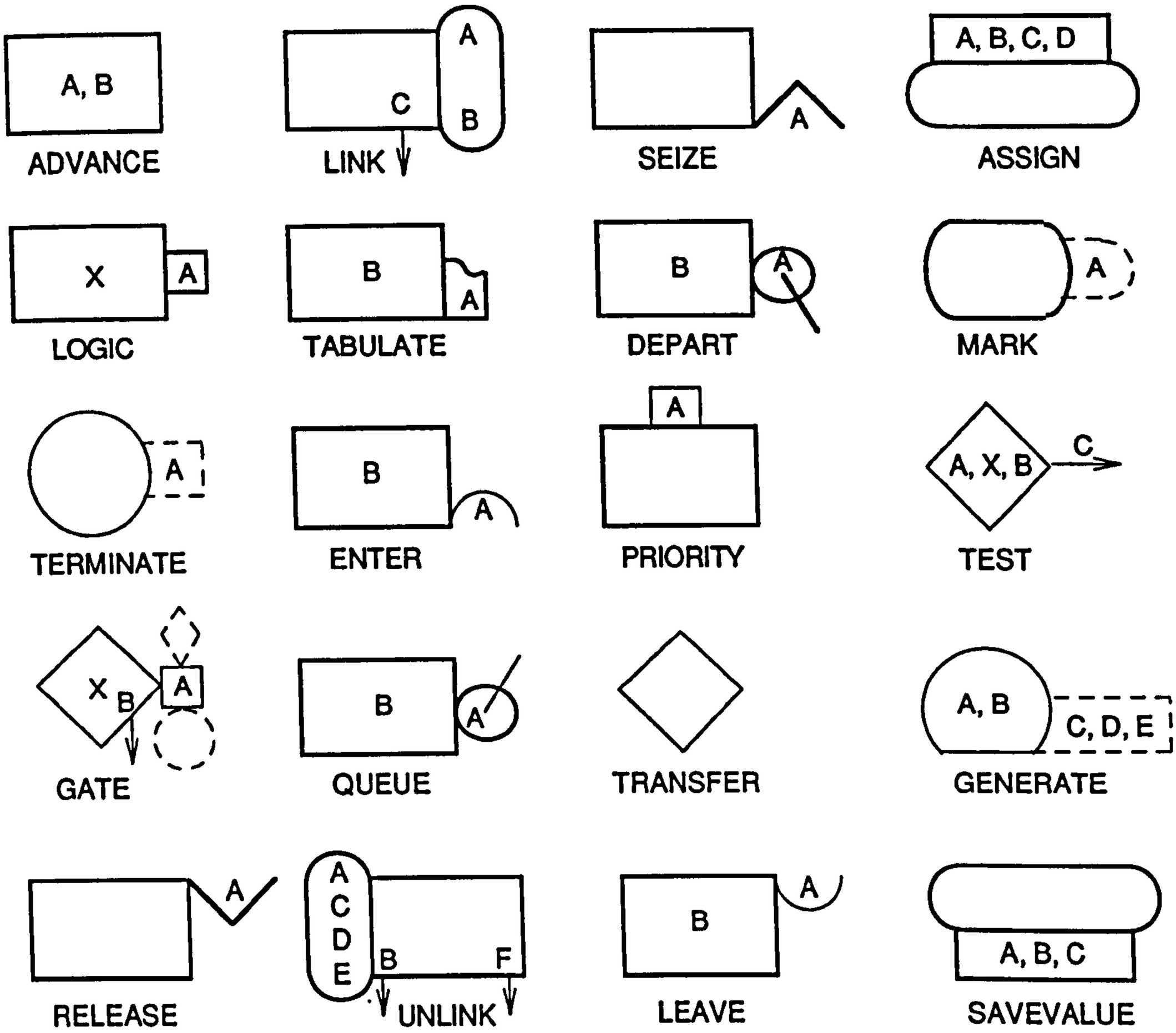


Figure 3.6 GPSS block-diagram symbols [Gordon 1978]

To each transaction is automatically assigned, by GPSS, a record with attribute fields called parameters. These parameters store numerical values which may be altered, during simulation, to reflect the change in the state of the model. Furthermore, GPSS uses the transaction record to store scheduling information. With this solution GPSS eliminates the need for temporary event notices or records associated with the entries in the list of scheduled events. This list is called, in GPSS, the future events list.

Transactions are active temporary entities which interact with each other and with passive permanent entities called facilities and storage. A facility is an entity that can be engaged by one transaction only at a given time. A storage is an entity that can be used by several transactions at a time, up to some predefined limit. Facilities and storage have

also attributes assigned implicitly by GPSS. Table 3.2 shows a list of some of the transactions, facilities, storage and system attributes called generically, by GPSS, Standard Numerical Attributes (SNAs).

Operation	A	B	C	D	E	F
ADVANCE	Mean	Modifier				
ASSIGN	Param.No.(±)	Source	(Funct.No.)	Param. type		
DEPART	Queue No.	(Units)				
ENTER	Storage No.	(Units)				
GATE	Item No.	(Next block B)				
GENERATE	Mean	Modifier	(Offset)	(Count)	(Priority)	
	(Params.)					
LEAVE	Storage No.	(Units)				
LINK	Chain No.	Order	(Next block B)			
LOGIC {R S I}		Switch No.				
MARK	(Param No.)					
PRIORITY	Priority					
QUEUE	Queue No.	(Units)				
RELEASE	FAcility No.					
SAVEVALUE		S.V. No. (±)		SNA		
SEIZE	Facility No.					
TABULATE	Table No.	(Units)				
TERMINATE		(Units)				
TEST	Arg. 1	Arg. 2	(Next block B)			
TRANSFER	Select. Factor	Next block A		Next block B		
UNLINK	Chain No.	Next block A		Count	(Param. No.)	
	(arg.)	(Next block B)				

( ) Indicates optional field

Table 3.1 GPSS Block Types [Gordon 1978]

GPSS model data can be modified by using assignment statements. The ASSIGN A,B statement (see table 3.1) enables the value of B to be assigned to the parameter A, where B can be any SNA (see table 3.2). The SAVEVALUE A,B statement allows the storage of any SNA to be used later. B stands for the SNA attribute to be saved and A defines the savevalue location to store the value of B. A savevalue is a global variable that can have different memory sizes and can store integer or floating point values. The PRIORITY statement provides an alternative way to set the priority of a transaction.

To handle facilities and storage GPSS uses the following statements: ENTER A,B; LEAVE A,B; SEIZE A; and RELEASE A (see table 3.1). The statement ENTER A,B enables a transaction to enter storage unit A, if it is available, and increase its contents by B. The statement LEAVE A,B does the opposite operation allowing a transaction to

leave storage unit A and decrease its contents by B. Each storage unit and its corresponding capacity must be defined using the STORAGE statement (see table 3.3). A storage unit with capacity **n** can be compared with a queuing system with **n** servers.

C1	The current value of clock time.
CHn	The number of transactions on chain n.
Fn	The current status of facility number n. This variable is 1 if the facility is busy and 0 if not.
FNn	The value of function n. (The function value may be computed to have a fractional part but the SNA gives only the integral part, unrounded.)
Kn	The integer n (the notation n may also be used).
M1	The transit time of a transaction
Nn	The total number of transactions that have entered block n.
Pn	Parameter number n of a transaction.
Qn	The length of queue n.
Rn	The space remaining in storage n.
RNn	A computed random number having one of the values 1 through 999 with equal probability. (when the reference is made to provide the input for a function, the value is automatically scaled to the range 0 to 1). Eight different generators can be referenced by n=1,2,...,8.
Sn	The current occupancy of storage n.
Vn	The value of variable statement number n.
Wn	The number of transactions currently at block n.
Xn	The value of savevalue location n.

Table 3.2 GPSS Standard Numerical Attributes (SNAs) [Gordon 1978]

The SEIZE and RELEASE statements (see table 3.1) behave similarly to the ENTER and RELEASE statements, respectively, when the storage unit has single capacity (one server). In this case it is not necessary to use the STORAGE statement and the server is called a facility.

Location	Operation	A	B	C	D
	CLEAR				
	END				
Function No.	FUNCTION	Argument	{C D L} No. of points		
	INITIAL	Entity	Value		
	JOB				
	RESET				
	SIMULATE				
	START	Run Count	(NP)		
Storage No.	STORAGE	Capacity			
Table No.	TABLE	Argument	Lower limit	Interval	No. of intervals

Table 3.3 GPSS Control Statements [Gordon 1978]

When a transaction cannot move ahead to the ENTER or SEIZE blocks, because of lack of resources (servers or facilities), GPSS links the transaction to the current events chain and to a delay chain associated with the required resource. GPSS also change the transaction attribute called the scan indicator to the inactive state. A transaction can be in active state (under system control), inactive state (waiting in the current events or future events chain until certain conditions are met), passive state (waiting on user chain until another transaction reactivated it), or terminated state (removed from the system).

After having executed all the events scheduled for a given time GPSS scans the current events chain to detect which transactions can be reactivated. GPSS uses the scan indicator of each transaction to decide if it is necessary or not to check if the transaction can continue processing. When the simulation conditions allow the transaction to move, GPSS removes the transaction from the current events chain and the execution resumes for that transaction at the defined reactivation point. The GPSS statement ADVANCE A,B is used to delay a transaction for a period of time while it is in service. The time is drawn from a uniform distribution with range (A-B,A+B) or from a user defined table for cumulative probability distribution function (see [Gordon 1978]).

The GPSS statements QUEUE, DEPART, MARK and TABULATE are used to collect statistics during the simulation run. The statement QUEUE A,B adds B units to the contents of the queue A referred as QA (see table 3.2). The statement DEPART A,B does the opposite reducing the contents of the queue A by B units. These statements also accumulate the product of QA (queue length) and the length of time that the queue has kept the same QA value.

The statements MARK and TABULATE allows the collection of data about the time taken by a transaction moving through the system between different blocks. The statement MARK A records in parameter A the transaction's arrival time to the MARK block. When the transaction arrives to a following TABULATE block, GPSS computes the elapsed time between the transaction arrival to the MARK block and the current time. These statements require the use of the TABLE A,B,C,D control statement (see table 3.3).

GPSS provides several statements to control the execution flow of a transaction through the system. The statement TRANSFER A,B,C allows a transaction to move to location B or C depending on a random draw from a uniform distribution on the interval (0,1).

If the result is less than A the transaction moves to location C, otherwise it moves to location B. This statement can also be used to do unconditional transfers in the form TRANSFER ,B.

GPSS handles 'on' 'off' situations by the use of logic switches and special statements. The statement LOGIC A followed by S, R or I in column 14 allows the switch A to be set (on), reset (off) or inverted, respectively.

The statement GATE A,B has a special condition code beginning in column 13 (e.g. LS A - Logic switch A set; LR A - Logic switch A reset; etc.; see [Gordon 1978]). If A satisfies that condition the transaction enters the GATE block, otherwise it moves to location B. If field B is left blank then the transaction will enter the GATE block only when the condition is met.

The TEST A,B,C statement compares the values of two A,B standard numerical attributes (SNAs) using symbols, beginning in column 13, to represent the desired relationships (e.g. G - Greater than; GE Greater than or equal; etc.; see [Gordon 1978]). The result of the testing condition is similar to the GATE block. The transaction will enter the TEST block if the condition is true, otherwise it will move to the location C. If the field C is omitted the transaction will enter the TEST block only when the condition becomes true.

Blocked transactions are automatically entered and removed, by GPSS, from sets (queues) using the FIFO (First In First Out) discipline. GPSS allows, however, the definition of user chains and the use of more complex queuing disciplines.

The statement LINK A,B places the transaction on chain A using the queuing discipline B. The term B may be replaced by the words FIFO or LIFO (Last In First Out) corresponding to the queuing disciplines with the same name. The term B may be also replaced by Pn. In this case the transactions are ordered by ascending values of their parameter number n and a FIFO rule is used when the same value is encountered. When a transaction moves to a LINK block it will stop there until another transaction removes it from the list using the UNLINK block.

The statement UNLINK A,B,C,D,E,F removes at most C transactions from chain A. The parameter C can be an integer, the value of an SNA or the word ALL (all the transactions will be removed). These transactions continue their processing at location

B. The unlinked transactions are the ones for which parameter D has a value equal to E. If no transactions are unlinked the present transaction moves to location F or to the next block if field F is left blank.

As mentioned previously, GPSS uses the process interaction approach to model building (see 2.2.4.). The structure of the GPSS executive is similar to the one presented in the logic diagram for a process interaction executive (see figure 2.4).

GPSS executive advances simulation time to the execution time of the first transaction on the future events chain. After, GPSS removes all transactions, with that same execution time, from the future events chain and links them to the current events chain. The scan indicator of these transactions is also changed to active.

GPSS executive scans the current events chain and checks the scan indicator of each linked transaction. When GPSS finds a transaction whose scan indicator is active it attempts to continue execution of that transaction. If the transaction can progress it will move ahead until it is delayed. If the delay is due to an ADVANCE block the transaction is linked to the future events chain. Otherwise, if the transaction is blocked by other causes (e.g. SEIZE, ENTER, or GATE blocks) GPSS changes the transaction's scan indicator to inactive and link it to the current events chain. If no resources have been released GPSS continues its scan of the current events chain. Otherwise, it will start a complete rescan of the current events chain.

GPSS has statements to control certain aspects of the simulation program (see table 3.3). The SIMULATE statement enables the execution of the simulation program and should be the first input statement to GPSS. When a simulation run is finished, GPSS does not immediately destroy the model. GPSS search first for more input following the START statement. It is possible to change the model using GPSS control statements, such as the STORAGE statement, or add or modify existing blocks. The use of the START statement will instruct GPSS to begin the simulation again. A more detail description of the GPSS control statements can be found in [Gordon 1978].

In Appendix III.F the GPSS block diagram and the model file for a one-line, one-server queuing system is presented [Schriber 1987]. The problem consists of a manufacturing system where castings are sent to a drilling machine, where each casting is to have a hole drilled in it. A complete description of this problem can be found in [Schriber 1987].

GPSS is one of the earliest simulation languages and is also very popular. Schriber [Schriber 1987] has presented a list of books about GPSS. In most of the books about simulation is also possible to find an introduction to GPSS with examples of its use. [Gordon 1978], [Fishman 1978] and more recently [Law & Kelton 1982] and [Neelamkavil 1987] are good introductions to GPSS.

3.2.3.2. SIMulation ANalysis (SIMAN)

The SIMAN simulation language [Pedgen 1984], introduced in 1982, was developed by C. Dennis Pedgen a faculty member of the Pennsylvania State University. Pedgen was involved previously in the design and implementation of the SLAM language while at the University of Alabama in Huntsville. The SLAM language was a significant improvement on GASP IV and Q-GERT (it was based mainly on Q-GERT (see figure 3.2)). According to Pedgen [Pedgen 1984] SIMAN was created to overcome some limitations of the SLAM language, especially in modelling certain manufacturing systems, and to include different simulation language concepts. These concepts had origin in Pedgen own experience and in the work done by Zeigler [Zeigler 1976].

Based on the concepts developed by Zeigler [Zeigler 1976] the SIMAN modelling framework makes a fundamental distinction between the system model and the experimental frame. The system model defines the characteristics of the system and the experimental frame defines the data under which the model is run. This approach allows the run of different simulation experiments by changing only the experimental frame. For a given system model and experimental frame the SIMAN simulation program generates an output file which records the model evolution through simulated time. The output data can then be analysed and used to display tables, histograms, etc.

Although SIMAN allows the event approach to be used, its primary modelling orientation is the process interaction (see 2.2.4.). As in GPSS, a SIMAN model is constructed by using blocks to build diagrams which represent the flow of entities through the system. The block diagram can be specified either graphically, in an interactive mode, or by statements in a batch mode. In SIMAN the event approach is implemented through user-written FORTRAN subroutines, describing the events, and a SIMAN library with subprograms that perform the standard modelling functions.

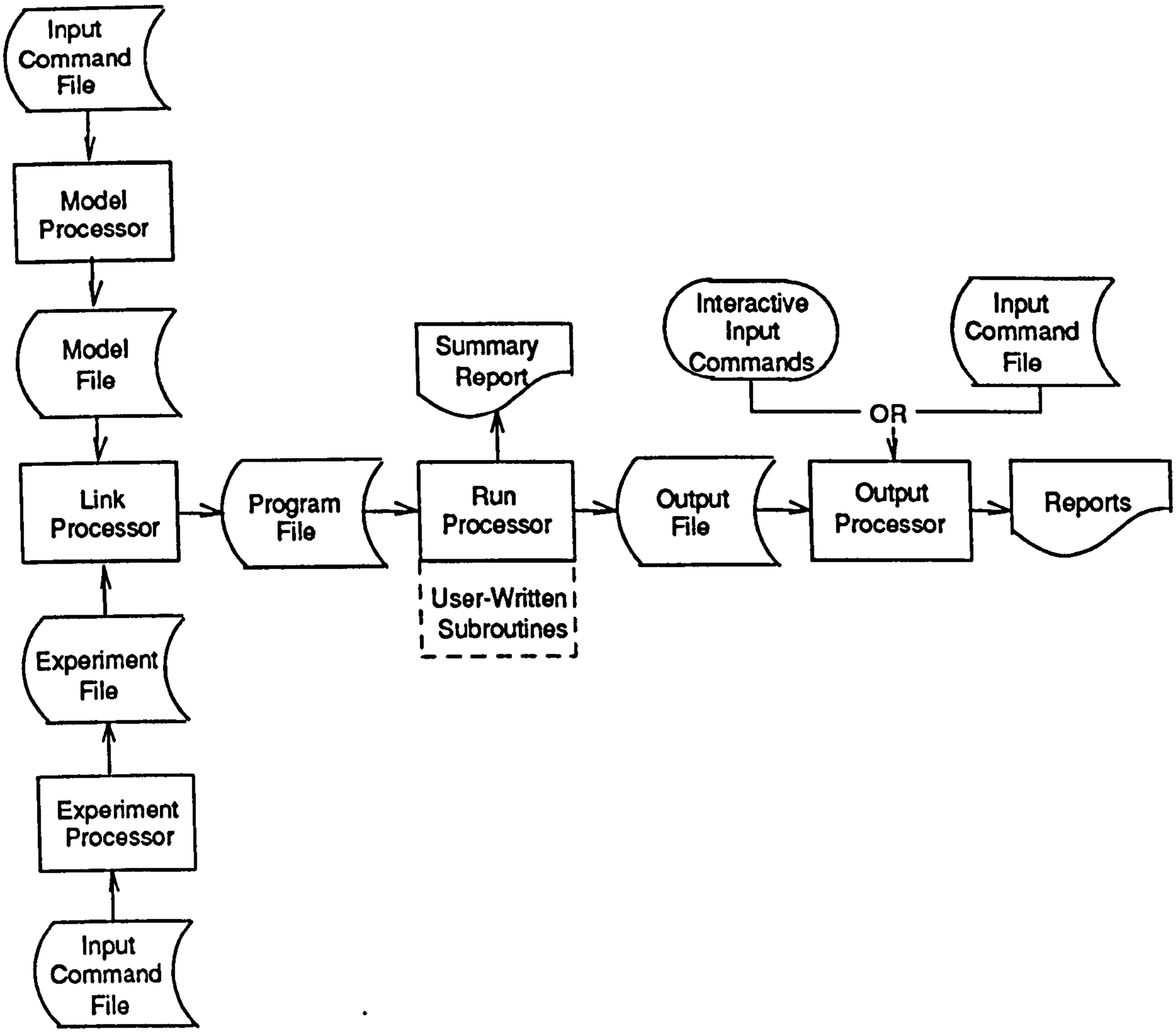


Figure 3.7 SIMAN software structure [Pedgen 1984]

A SIMAN simulation project consists of three distinct activities [Pedgen 1984]: system model development, experimental frame development, and data analysis. The SIMAN software enables the implementation of these activities using a structure, described in figure 3.7, which is based on the following processors [Pedgen 1984]:

model processor - builds the block diagram and generates the model data file;

experiment processor - defines the experimental frame and generates the experiment file;

link processor - combines the model file and the experiment file and produces the program file;

run processor - executes the simulation runs and writes the results on the output file;

output processor - analyses, formats and displays the data contained in the output file.

The block diagrams in SIMAN are linear top-down flowgraphs which describe the movement of entities through the system. The shape of each block indicate its function and arrows are used to represent the flow of entities through the diagram. Depending on the function of each block entities may be delayed, destroyed, combined with other entities, etc. Each entity may have associated attributes which are used to store its individual characteristics. SIMAN uses a real array A() for each entity to record its general purpose attributes during the simulation run.

The concept of using blocks to build simulation models in SIMAN is similar to the one used by GPSS (see 3.2.3.1.). In GPSS each block performs only one function and to both, block and function, is given the same name. The same principle is used by SIMAN except for the OPERATION, HOLD and TRANSFER blocks which can perform several different functions. In this case the block function name is the operation type, hold type or transfer type which is specified as the first operand of the block. The use of more then one function per block allows a significant reduction in the number of different block types required by SIMAN when compared with GPSS. SIMAN has only ten different basic block types whose symbols are presented in figure 3.8.

The blocks from figure 3.8 perform the following functions [Pedgen 1984]:

OPERATION - models a wide range of processes such as time delays, attribute assignments, etc. (see table 3.4);

TRANSFER - models transfers between stations via material handling systems (see table 3.5);

HOLD - models situations in which the movement of an entity is delayed based on system status (see table 3.6);

QUEUE - provides a waiting space for entities which are delayed at following HOLD or MATCH blocks;

STATION - defines the interface points between model segments and the material handling systems;

BRANCH - models the conditional, probabilistic and deterministic branching of entities;

PICKQ - selects from a set of following QUEUE blocks;

SELECT - selects between resources associated with a set of following OPERATION blocks;

QPICK - selects from a set of preceding QUEUE blocks;

MATCH - delays entities in a set of preceding QUEUE blocks until entities with the same value of a specified attribute resides in each QUEUE.

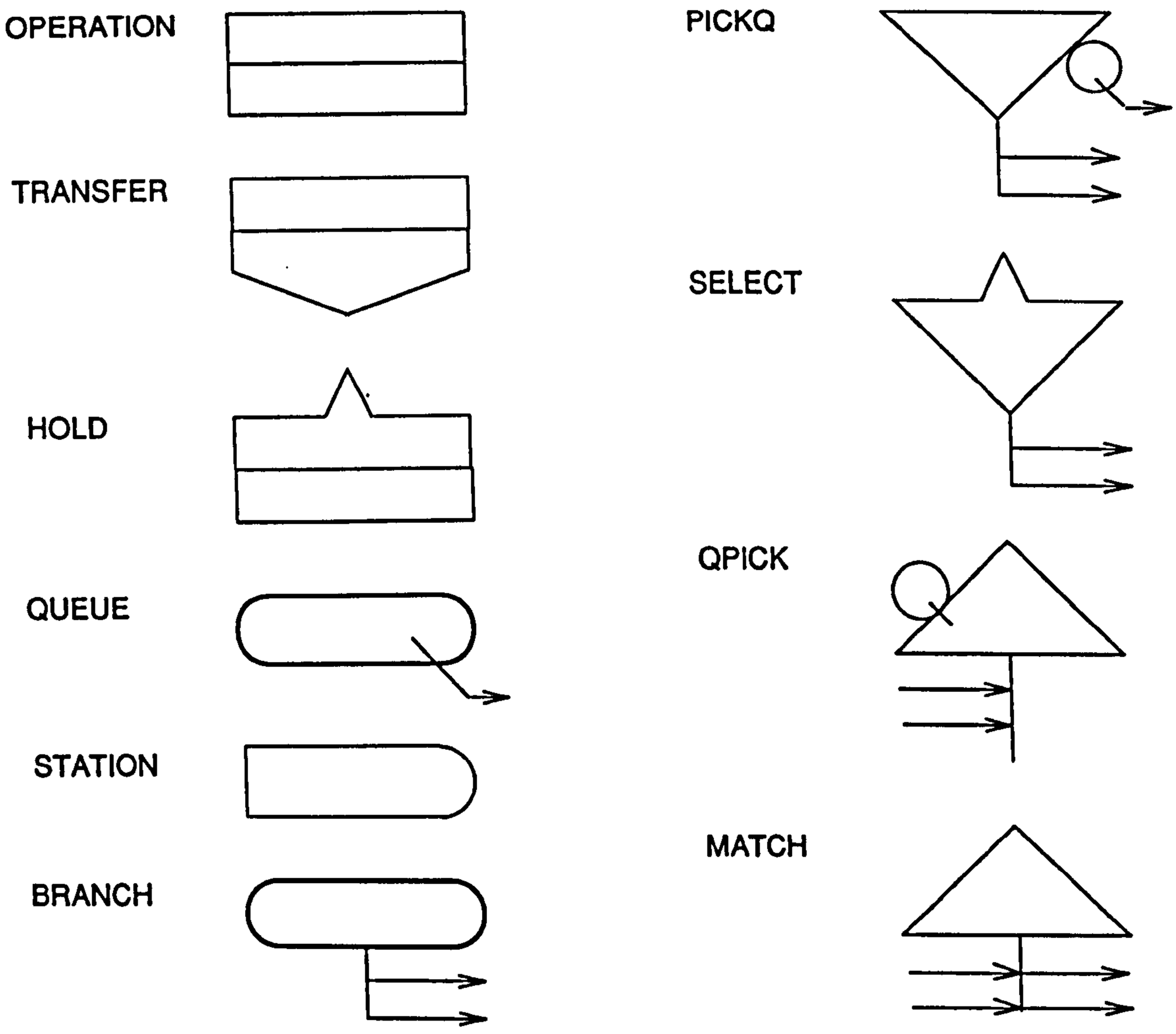


Figure 3.8 SIMAN basic block types [Pedgen 1984]

The several different functions that may be used with the blocks OPERATION, HOLD and TRANSFER are listed in tables 3.4 to 3.6. The function type is specified as the first operand of the block and is common practice to address the block by the name of the function it performs. For example, the ASSIGN block is used to assign values to attributes or variables and the DELAY block is used to delay an entity by a certain period of time (see table 3.4). All the blocks have operands which specify the parameters for the function they perform. For example, in the CREATE block it is necessary to specify the time between batch arrivals, the number of entities per batch, and the maximum number of batches to create.

	Name	Description
General Functions	ASSIGN	Assign values to attributes and variables.
	CREATE	Create batch arrivals to the system.
	DELAY	Delay an entity by a specified time.
	DETECT	Detect state events associated with continuous variables.
	EVENT	Cause a specified event to occur.
	FINDJ	Find the value of the index J meeting a specified condition.
	SIGNAL	Send a signal to end the delay for an entity.
	SPLIT	Split a group into its individual members.
Resource Functions	ALTER	Change the capacity of a specified resource.
	RELEASE	Release units of a specified resource.
Material Handling Functions	ACTIVATE	Set the status of a specified transporter to active.
	EXIT	Exit a specified conveyor device.
	FREE	Exit a specified transporter device.
	HALT	Set the status of a specified transporter to inactive.
	START	Set the status of a specified conveyor to active.
	STOP	Set the status of a specified conveyor to inactive.
File Functions	COPY	Copy the attributes of an entity at a specified QUEUE.
	REMOVE	Remove an entity from a specified QUEUE.
	SEARCH	Search a QUEUE for an entity meeting a specified condition.
Statistical Functions	COUNT	Increment a specified counter.
	TALLY	Record an observation of a specified value.

Table 3.4 SIMAN Operation Types [Pedgen 1984]

When constructing a block diagram it is possible to assign to each block a label, one or more modifiers, and a comment line. The label can have up to eight alphanumeric characters, placed on the lower left side of the block, and is used for branching or

referencing from other blocks. The block modifiers are symbols, placed on the right or bottom of the block, that are used to modify or extend the function performed by the block. The comment line, placed on the right of the block, is used to document the block diagram.

	Name	Description
Condition Functions	SCAN	Hold the entity until a specified condition is met.
	WAIT	Hold the entity until a specified signal is received from a SIGNAL block.
Resource Functions	SEIZE	Hold the entity until the required number of units of a resource are idle and are allocated to the entity.
	PREEMPT	Hold the entity until one resource unit is allocated to the entity. The entity may preempt a resource currently being used.
Material Handling Functions	ACCESS	Hold the entity until a specified number of consecutive conveyor cells are available and allocated to the entity at the accessing station location.
	REQUEST	Hold the entity until a transporter device is allocated to the entity and arrives to the requesting station location.
Set Functions	COMBINE	Hold the entity until a specified number of entities reside in the preceding QUEUE block. When this occurs, the waiting entities are combined into a permanent set and a representative of the set is created. The original entities in the set are destroyed.
	GROUP	Hold the entity until a specified number of entities reside in the preceding QUEUE block. When this occurs, the entities are grouped into a temporary set and a representative of the set is created. The original entities in the set are retained and can be recovered using the SPLIT block.

Table 3.5 Hold Types [Pedgen 1984]

Figure 3.9 describes the SIMAN block modifiers. The connector symbols are used to determine the flow of the entity through the system and are appended to the bottom of the block. The sequential flow connector (figure 3.9 a)) is used for directing sequential flow of entities from one block to the next. The non-sequential flow connector (figure 3.9 b)) is used for directing the flow of entities to the block labelled with the specified LABEL. If neither connector is used the symbol from figure 3.9 c) is added to indicate that the entities leaving the block are destroyed. The mark symbol from figure 3.9 d)

may be placed on the right side of the block and is used to mark the attribute number MA with the arrival time of the entity to the block.

Name		Description
Material Handling Functions	CONVEY	Convey the entity to a specified station via a conveyor. The transmit time is determined by the distance between the station along the conveyor and the speed of the conveyor.
	ROUTE	Route the entity to a specified station. The transmit time is specified as an operand of the block.
	TRANSPORT	Transport the entity to a specified station via a transporter. The transmit time is proportional to the distance between stations.

Table 3.6 Transfer Types [Pedgen 1984]

In Appendix III.G the SIMAN block diagram, the input statements and the corresponding experimental frame for a TV inspection system are presented [Pedgen 1984]. In this system, the vertical control setting of TV sets is tested at an inspection station. If a TV set is found defective it is routed to an adjustment station. After adjustment, the TV set is sent back to the inspection station where it is again inspected. TV sets passing the inspection are sent to a packing area. A complete description of this problem can be found in [Pedgen 1984].

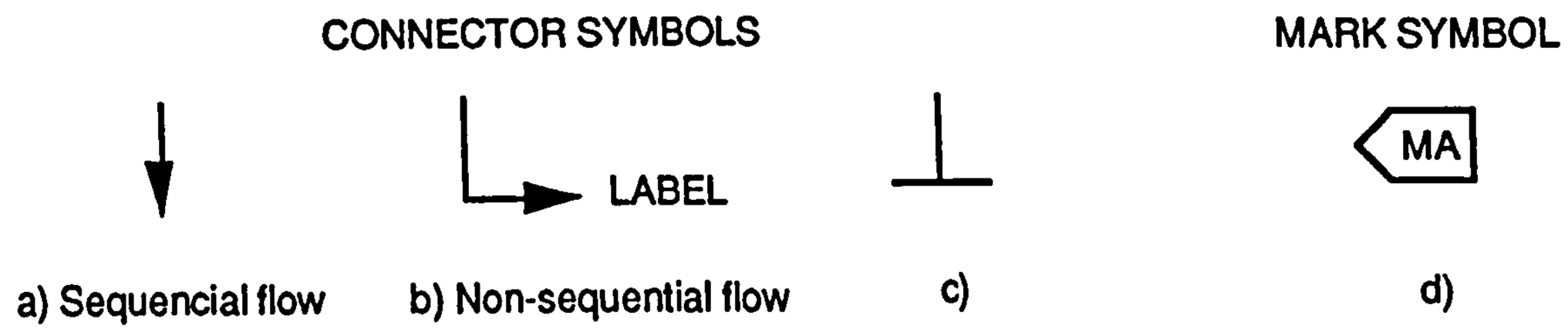


Figure 3.9 SIMAN Block Modifiers [Pedgen 1984]

One of the important SIMAN's features consists of a set of special purpose constructs oriented to the modelling of manufacturing systems. These systems present certain

characteristics that may not be modelled accurately or require a great modelling effort when using a general purpose simulation language. These characteristics were identified by Pedgen [Pedgen 1986] who pointed out that this was particularly relevant when the manufacturing system had a major material handling component.

SIMAN has the ability to model distinct workcenters, in a manufacturing system, by using the STATION block (see figure 3.8). This block defines the beginning of a station submodel which describes the work done at the workcenter. The TRANSFER block (see figure 3.8) is used to move entities between station submodels. SIMAN can handle different types of material handling equipment which are classified, for modelling purposes, as transporters and conveyors.

SIMAN uses the term transporter to group devices that perform what is called the transport function. This function consists [Pedgen 1986] of the intermittent movement of items, one load at a time, along a fixed or varied path. From a SIMAN modelling perspective transporters can be material handling equipment such as industrial trucks (forklift trucks, hand carts, etc.), cranes, hoists, and manipulators.

An entity allocates a transporter unit at HOLD blocks (see figure 3.8) using the REQUEST hold type (see table 3.5). The TRANSFER block (see figure 3.8) with the TRANSPORT transfer type (see table 3.6) is then used to move the entity from one station to the next. SIMAN computes automatically the duration of the transport based on the distance between the two stations and on the speed of the transporter unit. In the end of the transport the transport unit may be released using the OPERATION block (see figure 3.8) with the FREE operation type (see table 3.4).

SIMAN uses the term conveyor to group devices that perform which is called the convey function. This function consists [Pedgen 1986] in the continuous and simultaneous movement of items along a fixed path. From a SIMAN modelling perspective conveyors can be material handling equipment such as belt conveyors, bucket conveyors, hook conveyors, and trolley conveyors.

An entity allocates a conveyor cell at HOLD blocks (see figure 3.8) with the ACCESS hold type (see table 3.5). The TRANSFER block (see figure 3.8) with the CONVEY transfer type (see table 3.6) is then used to move the entity to its destination workcenter. The OPERATION block (see figure 3.8) with the START and STOP operation types

(see table 3.4) can be used to start or stop a conveyor. A detailed description of the SIMAN manufacturing systems special constructs can be found in [Pedgen 1984].

Graphics and animation are available to SIMAN models through the use of the package CINEMA. CINEMA [Kilgore & Healy 1987] is a general purpose animation system designed to work with the SIMAN simulation language. With CINEMA it is possible to produce a dynamic display of graphical objects that change size, shape, colour or location on a static background in correspondence with changes in a concurrently executing SIMAN simulation model [Kilgore & Healy 1987]. CINEMA requires no programming and has a user friendly interface. This interface uses "pulls-down" menus accessed with a mouse which is also used as a drawing device.

The CINEMA animation layout is composed of a static background and dynamic graphic objects. The static background is created using a set of elementary drawing functions or using a special facility to import AutoCAD drawings. The dynamic objects are associated with SIMAN model specific elements and are automatically updated reflecting the model changes during the simulation. CINEMA has facilities to create graphic symbols (icons) that can be maintained in symbol libraries. These symbols can be linked to simulation model elements to display their status during the simulation run. Examples of these are the entity, resource and transporter symbols.

The entity symbol is used to show during the animation the movements and changes of an entity. The symbol is linked to the entity by a general purpose entity attribute that must be reserved for this purpose in the SIMAN model. The symbol can be changed just by assigning a new value to the entity attribute. This allows different stages, during the progress of the entity, to be represented by different symbols.

The resource symbols are used to represent the four possible states a resource can assume: idle, busy, inactive and preempted. The symbol is associated with the resource through the SIMAN resource number. This implies that the dynamic symbols must be created after the creation of the SIMAN model.

The transporter symbols are used to represent the three possible states a transporter can assume: idle, busy and inactive. The symbol is linked to the transporter through the SIMAN transporter number. The transporter symbol is also used to show the movement of the transporter between stations.

The movement of entities and transporters between stations is handled in SIMAN by the TRANSFER block. CINEMA uses a TRANSFER section to group the tasks concerning the definition of the screen paths for moving the entities between stations. After all the stations symbols have been positioned the path between them is defined through a series of connected line segments. The station symbols are numbered with the corresponding station number in the SIMAN model. When an entity is transferred between two stations its symbol is displayed along the corresponding path. If a transporter is used its busy symbol is also shown moving along the path.

CINEMA uses a special queue symbol to enable the display of the entities waiting in a queue. This symbol consists of a line segment of any length or orientation that can be placed anywhere in the layout. The queue symbol is linked with a specific file number associated with a SIMAN QUEUE block. As the entities join the queue their symbols are represented along the line segment. The head of the queue corresponds to the first point defined on the line segment.

CINEMA provides also additional facilities to display the value of any SIMAN status variable during the execution of simulation. A CINEMA tutorial introduction can be found in [Healy 1986] and [Kilgore & Healy 1987]. A full description of the SIMAN simulation language can be found in [Pedgen 1984] and a brief introduction in [Pedgen 1986].

3.2.4. Program Generators

A program generator was defined by Mathewson [Mathewson 1989] as "interactive software tools that translates the logic of a model, described in a relatively general symbolism, into the code of a simulation language or the appropriate list of building block definitions". Program generators have been available since the beginning of the 1970s. Their objective is mainly to aid the user in the task of writing the code for the simulation model. This assumes a particular importance when the user has little experience in computer practices.

Program generators work usually in computer interactive environments and are also called Interactive Program Generators (IPGs) [Pidd 1988]. An interactive dialogue between the user and the IPG is used to define the logic of the model. To be able to answer the questions in this dialogue the user must first create an activity cycle diagram

for the system he wants to simulate. Based on the information contained in the activity cycle diagram, described by the user, the IPG is able to generate the simulation source code in an appropriate simulation language.

The program generator can successfully replace the host language in modelling a large number of different systems. However, specially when the system to be model is complex, it may be necessary to do some further programming editing the result source code. Although, the source code produced by IPGs is usually well documented, the user can face some difficulties if he is not an experienced programmer. Another difficulty that the less experienced user may find concerns the error messages, given by the IPG, reporting errors in the generated code. This is due to the fact that those messages being related to the target simulation language will force the user to have a deeper involvement with the generated source code.

Two of the best-known Interactive Program Generators are CAPS [Clementson 1986] and DRAFT [Mathewson 1985]. CAPS generates ECSL code and DRAFT is available in several versions which can produce code in SIMON, GASP II, SIMULA or SIMSCRIPT.

3.2.4.1. Computer Aided Programming System (CAPS)

CAPS [Clementson 1986] uses a conversational mode which allows a user to define the simulation model by his responses. Based on the logic specified by the user, CAPS generates an ECSL program. Normally, this program will correspond to a valid model and will be compiled and executed without any errors. However, the user must be careful to make sure that the generated code corresponds to his idealized model.

Before starting a CAPS session the user must prepare the required information to define the simulation model. Most of this information is obtained from the activity cycle diagram that should be drawn in advance. Clementson [Clementson 1986] has presented the following as a check list of the things that should be prepared:

- a detailed activity cycle;

- the maximum number of each type (class) of entities to be used in the simulation;

- optionally, the queue disciplines for any queues which are not FIFO (First In First Out);

- the formula for the duration of each activity;

- an idea of which class sizes are likely to be the subject of experiment and the range of values which may be required;

- a decision as to which queues are to be recorded and in which way (length of queue, delay time, utilization);

- the chosen length of the simulation including any run-in period;

- an initial state, expressed as activities in progress and entities in queues.

After putting together all the described information the user is ready to start the CAPS session. This session takes the user through a sequence of stages that are described in figure 3.10 grouped in the major phases of the work. In case something needs to be corrected the user has the opportunity, at the end of each stage, to go back to a previous section. The CAPS session ends with the generation of the ECSL program corresponding to the described model.

The activity cycle diagram and the CAPS dialogue [Pidd 1988] used to generate the ESCL program from Appendix III.D (see 3.2.2.1.) are presented in Appendix III.H. This example shows the straight forward process of generating the ECSL program using CAPS. Although having a very user friendly interface CAPS requires the user to have a reasonable knowledge about simulation. This is necessary to create the activity cycle diagram and to answer CAPS technical questions. Moreover, as CAPS does not incorporate every ECSL features it may be necessary, in certain cases, to edit the generated program in order to enhance it.

3.2.4.2. DRAFT

DRAFT [Mathewson 1985] is an Interactive Program Generator (IPG) using a similar approach to CAPS but that can produce code in different simulation languages. DRAFT

uses an interactive dialogue, based on the activity cycle diagram, to enable the user to describe the simulation model. The DRAFT program generator is organized as a group of functional modules described in figure 3.11.

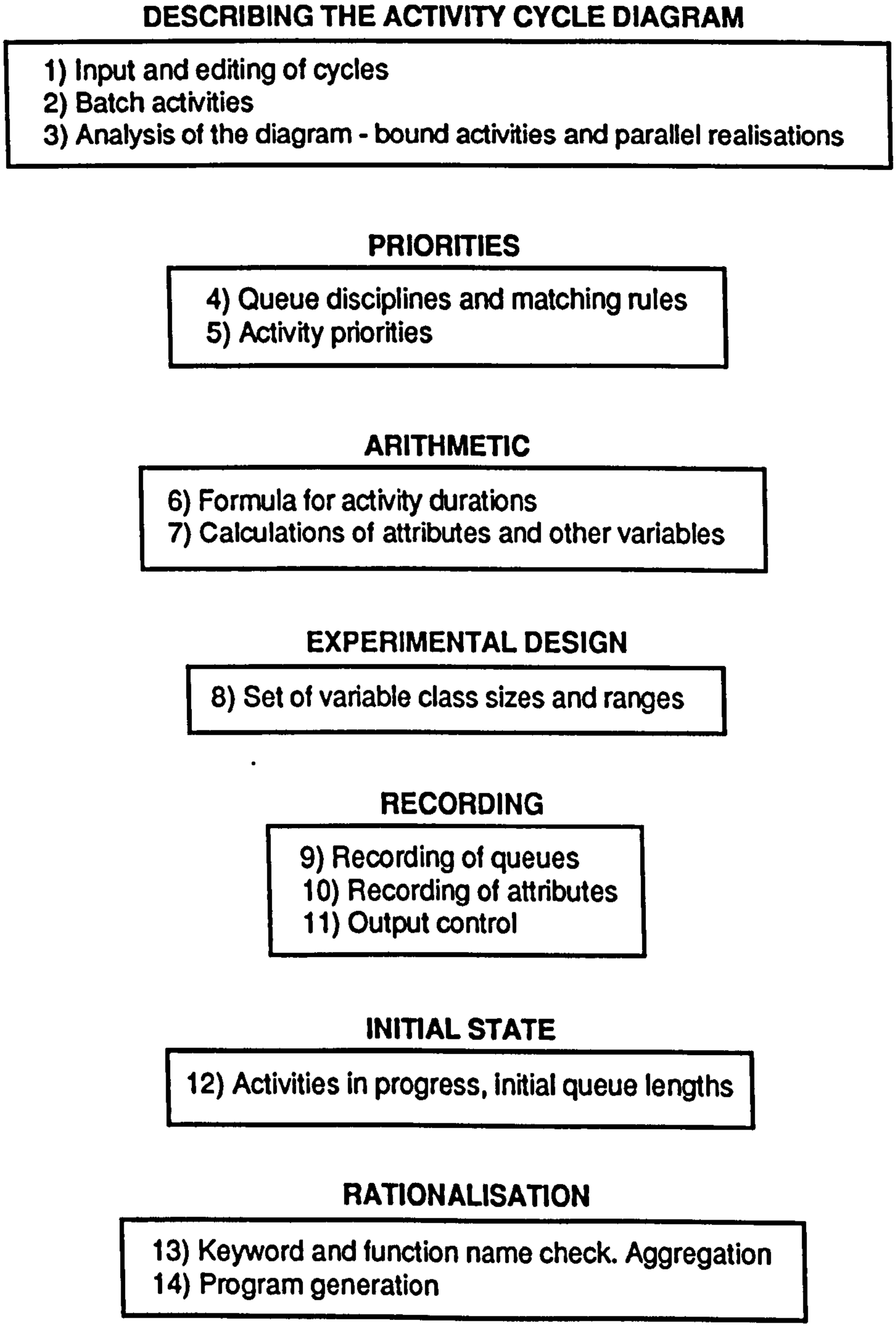


Figure 3.10 The Stages of Discussion of a CAPS session [Clementson 1986]

The input/editor module (see figure 3.11) allows the user to enter the model specification. This module has editing facilities and does some minor error checking (e.g. reserved words, duplicated names). The input of the interactive session is saved in 'SCRATCH' files (see figure 3.11) which can be modified later. The model specifications can also be saved in 'FILES' (see figure 3.11) in a compact form.

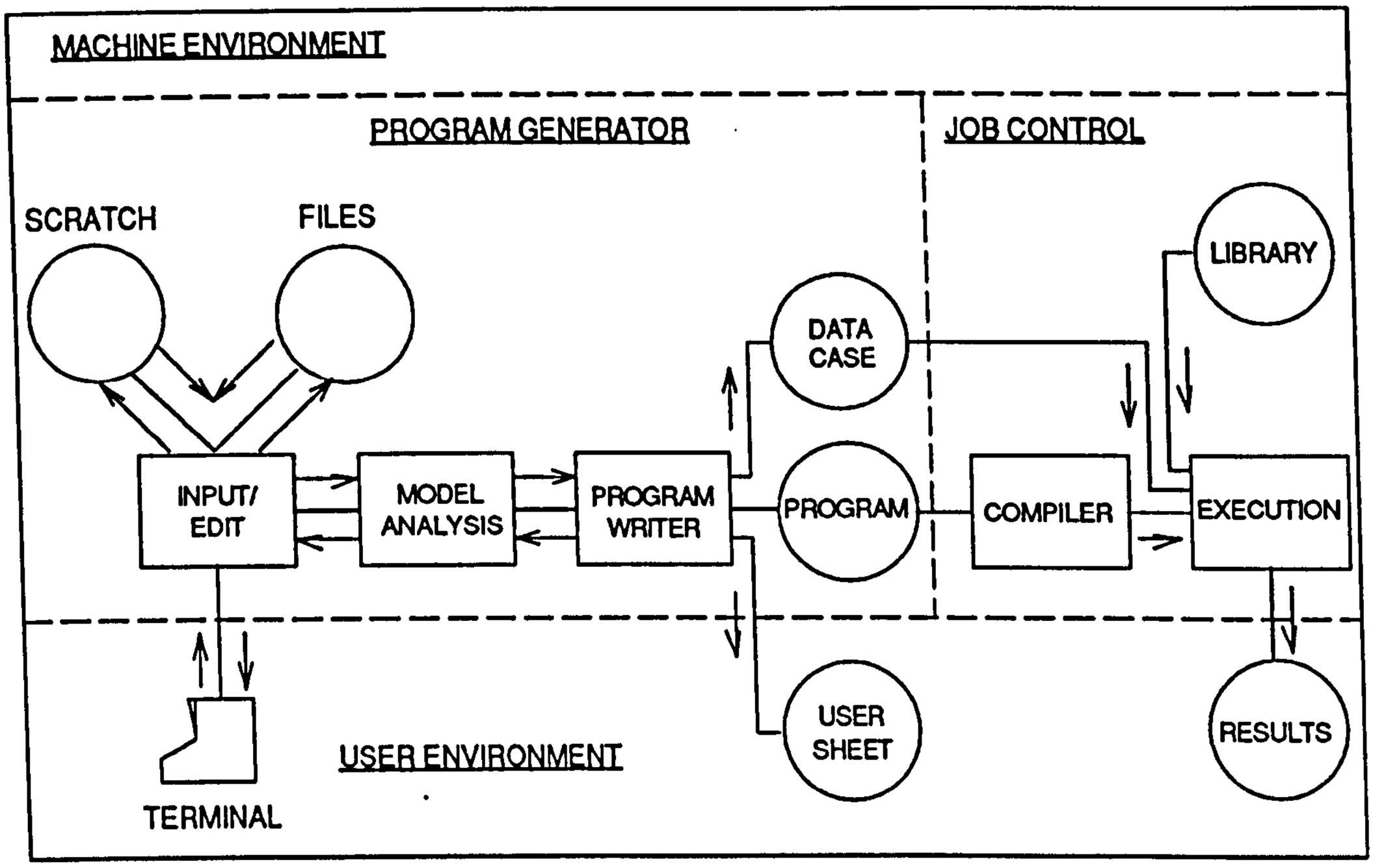


Figure 3.11 The Structure of DRAFT Program Generator [Mathewson 1985]

The analysis module (see figure 3.11) performs an error checking on the model specifications and produces an intermediate coded file to be used by the program writer. If errors are found they can often be corrected on-line. The program writer (see figure 3.11) generates the program code in the selected language. It is at this stage that the simulation model structure is created. The event, activity or process approaches are used depending on the target language.

The activity cycle diagram and the DRAFT session [Mathewson 1989] used for the development of a simulation model of a robot cell are presented in Appendix III.I. The system to be modelled is a small manufacturing unit consisting of a numerically controlled mill, with two work tables each holding an identical jig. The machine is

serviced by a robot. A complete description of this example can be found in [Mathewson 1989] as well as the listing of the generated FORTRAN program.

3.2.5. Data-Driven Generic Models

Someone wanting to build a simulation model using one of the approaches already described must have a reasonable knowledge about simulation. This is because the model is defined using simulation terminology (e.g. entities, queues, activities, etc.) and a logic (e.g. event-based, activity-based, etc.) that depends on the simulation approach used by the simulation system.

The data-driven generic model approach is oriented to specific areas and hence can use a terminology and a logic that can be easily understood by the end user. The data-driven generic model concept has a generic model that covers a wide range of similar problems and each particular model is defined by specifying data to the generic model. The simulation systems using this approach are also known as application-specific systems, generic models or simulators.

Manufacturing systems is an area where several data-driven generic models have been developed in the last decade. These simulation systems have taken full advantage of computer interactive and graphical facilities, especially microcomputers. Most of these simulation systems have user friendly interfaces, use interactive colour graphics and allow the model definition to be made by someone with minimum knowledge about simulation. Two good examples are WITNESS [Gilman & Watremez 1986] and SIMFACTORY II.5 [CACI 1990].

3.2.5.1. WITNESS

WITNESS [Gilman & Watremez 1986] is a menu driven simulation program written in SEE-WHY (see 3.2.1.2) for the study of manufacturing systems. WITNESS uses Visual Interactive Simulation (VIS) (see 2.3) and a manufacturing terminology which makes simulation available to the non-specialist user.

A WITNESS model is defined in a single environment by the use of menus and prompts. There is no need for programming and no code is generated. The model is

built in three main phases: Define, Display and Detail. The user can go through these phases in any order or move, at any time, between them and the execution phase. After executing WITNESS the user is prompted for the model name, the title, the author name and the date. After this initial prompting the WITNESS's main menu is displayed and the following commands become available [ISTEL 1987]:

DEFINE - defines the names and quantities of the elements to be used in building the model;

DISPLAY - allows the specification of how the elements will be displayed on the screen;

DETAIL - allows the definition of the logic which determines how each element will operate;

INITIALIZE - enables the setting of the starting conditions for a particular run of the model;

GO - runs the simulation model;

KEEP - saves the model for future use;

LIST - produces a listing on the screen of the elements comprising the model;

DELETE - deletes elements from the model;

END - ends the WITNESS session without saving any changes;

The first three commands DEFINE, DISPLAY and DETAIL are used to build and modify the model where the others are used to operate on a model after it has been built. After leaving the main menu (e.g. the GO command is selected) the same commands along with others are displayed in a blue box in the top right hand corner of the screen. This box is called the Interact Box and is used to display the different levels of WITNESS's commands and for prompting the user for information.

WITNESS models are based in several simulation elements which interact in specified ways. These elements are classified as physical elements and control elements. The physical elements are used to represent the manufacturing plant and the control elements are used to control and monitor the plant. The following are the physical elements from WITNESS version 3 [ISTEL 1987]:

Parts - are the individual elements (entities) which flow through the model and are processed by the system being modelled (e.g. physical components, pallets, etc.);

Buffers - are storage areas where Parts can be held (e.g. people in a queue, pallets on a shelf, etc.);

Machines - are powerful elements which are used to represent anything that takes Parts from somewhere, processes them and sends them on to their next destination (a drill press, a plant, etc.);

Conveyors - are transportation devices where all Parts being carried move in unison (belt conveyors, roll conveyors, etc.);

Vehicles - represent Automatic Guided Vehicles or other devices whose purpose is to transport parts;

Tracks - are the paths that Vehicles follow when transporting Parts;

Labour - is a resource pool which may be required by other elements as they perform their own respective operations.

The following are the control elements from WITNESS version 3 [ISTEL 1987]:

Attributes - are individual characteristics associated with Parts (e.g. part numbers, process times, etc.);

Variables - are variables which can store decimal or whole number values and can be accessed from anywhere in the model;

Distributions - are standard distributions to help to introduce variability into a model (e.g. uniform, exponential, Poisson, etc.).

The physical and control elements are created in the Define phase using the DEFINE command. For each element it is necessary to specify a name, the quantity and, in some cases, the type. When creating a model it is necessary first to define its elements and only then the DISPLAY and DETAIL commands can be used in any order.

The Display phase is used to specify how the WITNESS's elements should be displayed on the screen. After executing the DISPLAY command, WITNESS, from the interact box, will ask for the names of the elements and the user is then able to define the colour, shape and position of each one. Eight colours can be used for the foreground and background. The elements can be positioned on any of four windows but only one window can be viewed at any one time. Each window is 80 characters wide by 48 characters high.

Parts are all displayed in the same way as a count or as filled rectangles one character high by 1-4 characters wide. The Parts display position is defined by the element where they are located. Machines are displayed as shapes chosen from a WITNESS icon library. Conveyors are represented as a row or column where the gaps between the Parts are shown by a user chosen pattern. Buffers are also represented as a row or column or as a number indicating the number of Parts in the Buffer. Labour is displayed as a 1-4 character square. Tracks are represented in the same way as conveyors. Vehicles are displayed by the Tracks or by others elements they are on.

The Detail phase is used to specify times in the model, flow of Parts, Labor requirements, breakdowns and setups. WITNESS uses a series of Detail Forms, a different one for each element type, to enable the specification of this information. Most of the parameters of each element assume default values except those which describe Part routing. The flow of Parts through the system is determined by input and output rules. Machines, Conveyors, Vehicles and Buffers all have associated input and output rules. The function of these rules is basically to push and pull Parts through the different elements in the system.

A push output rule will cause a Part to be pushed to an element (e.g. Machine, Conveyor, Vehicle or Buffer) if it is prepared to receive it otherwise the element pushing will assume a blocked state. A pull input rule will cause an element to only receive input when it needs it instead of receiving input when the previous operation is ready to provide it. WITNESS has several inputs (e.g. WAIT, PULL, MOST, LEAST, %, SEQUENCE, IF, SELECT, BUFFER) and output rules (e.g. WAIT, PUSH,

MOST, LEAST, %, SEQUENCE, IF, SELECT, PART, BUFFER, DESTINATION, VEHICLE) which gives the user the ability to model complex routing.

WITNESS input and output rules can be used selectively in conjunction with the Actions feature. Actions are a series of computations steps that change the values of variables and attributes, evaluate expressions, change Part description or colours, and perform many functions that are needed to model Tracks and Vehicles. These actions take place at the entry to or departure from some component of the system. The Actions can be created or modified during the Detail phase using a simple editor.

WITNESS provides several system functions that can be used wherever numeric expressions are required such as in Detail Forms or in Actions. For example, NPARTS will return the number of Parts in the machine buffer or conveyor specified. WITNESS also offers several system Variables and system Attributes. For example, TIME is a system variable that holds the current simulation clock time. The system Attributes can be used to change Part descriptions or Part colours.

WITNESS also collects statistics and produces automatically reports, when selected in the Detail Form, for the different elements in the system (e.g. average size of a buffer, percent of time a conveyor was blocked, etc.). A special WITNESS facility allows the storage of models in files, known as library files, that can be edited to modify the model or used to create new models. When WITNESS is not capable of handling certain particularities of a model, SEE-WHY (see 3.2.1.2.) sub-models can be built and linked with the WITNESS model.

WITNESS allows people with no simulation experience to develop simulation models of manufacturing systems in a user-friendly environment. The use of Visual Interactive Simulation (VIS) in conjunction with a model building user interface that uses manufacturing terminology makes WITNESS very attractive to the less experienced user. An introduction to WITNESS is given in [Gilman & Watremez 1986] and a full description can be found in WITNESS manual [ISTEL 1987].

3.2.5.2. SIMFACTORY

SIMFACTORY [Kleine 1986] is a simulation program written in SIMSCRIPT II.5 (see 3.2.2.2) for the study of manufacturing systems. As does WITNESS, SIMFACTORY

makes use of Visual Interactive Simulation (VIS) (see 2.3), menus, interactive colour graphics and a manufacturing terminology to make simulation easier and more accessible to non-programmers.

SIMFACTORY II.5 [CACI 1990] is the latest version of SIMFACTORY and runs on IBM PC compatible microcomputers and on some workstations. SIMFACTORY enables the building of a factory model using a mouse, menu bars, buttons, and other graphic concepts and requires no programming. The model layout can be easily created using the Layout Editor. Although SIMFACTORY provides pre-defined icons, an Icon Editor can be used to create or modify icons representing the different elements in the model.

The model is created using basic "building blocks" (e.g. Stations, Buffers, Transporters, etc.) and by selecting their order, content, and parameters (e.g. speed, capacity, etc.). Figure 3.12 list the SIMFACTORY pull-down menus in the layout editor. A brief description of the different options is presented.

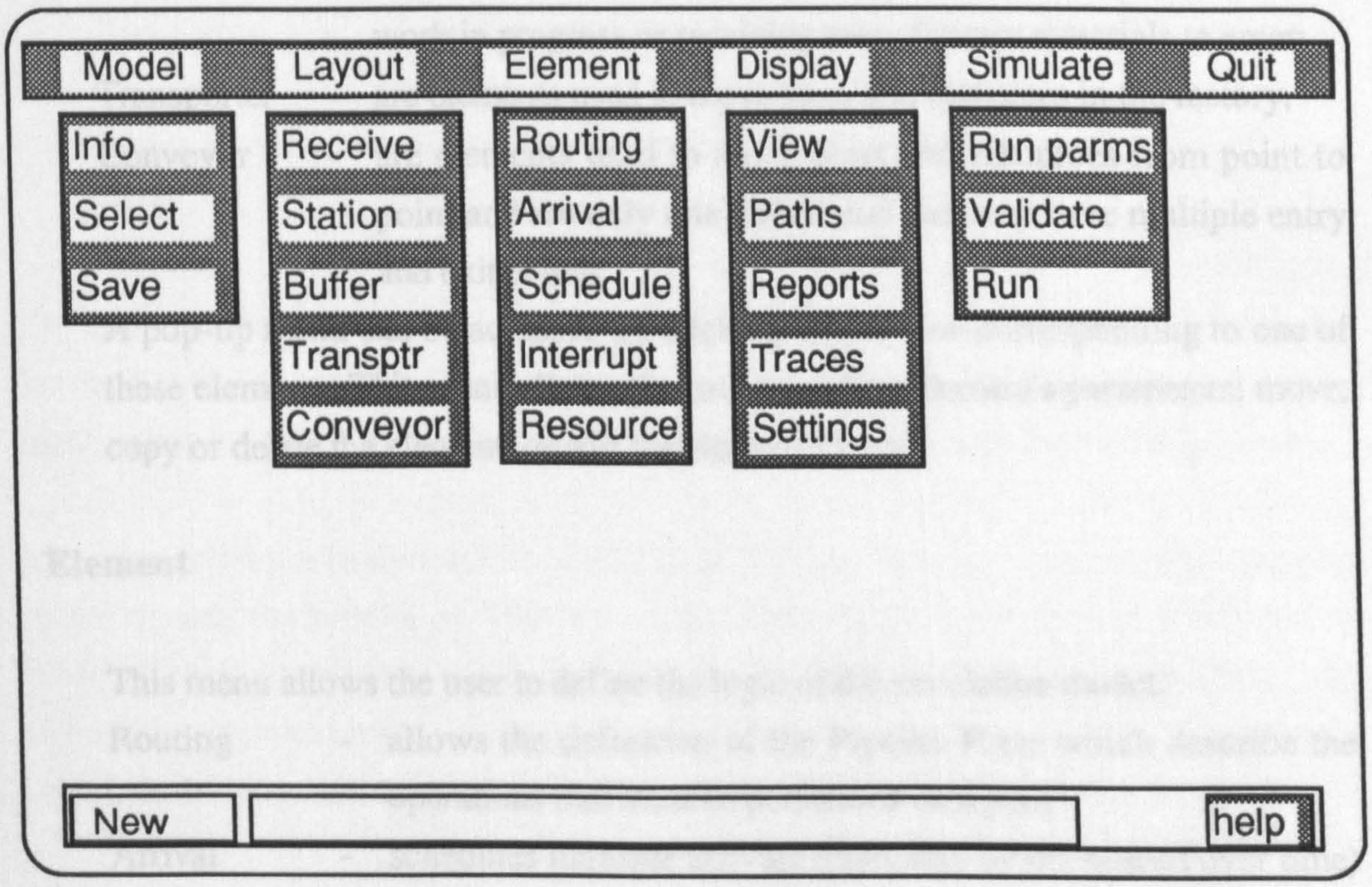


Figure 3.12 The SIMFACTORI II.5 main menus [CACI 1990]

. Model

This menu allows the user to define system parameters and to save and retrieve models.

Info - allows the modification of system parameters (e.g. as time, distance units, etc.);

Select - allows the selection or deletion of an existing model or the creation of a new one;

Save - saves the current model under a chosen name.

. Layout

This menu is used to add any of the following physical elements to the model:

Receive - are elements that work as buffers to receive raw materials from outside the factory;

Station - are elements that perform the necessary operations to process a part;

Buffer - are elements that provide staging areas in the factory to store work in progress or receiving areas for raw materials to enter;

Transporter - are elements used to move parts and resources in the factory;

Conveyor - are elements used to move parts and resources from point to point and are only one directional and may have multiple entry and exit points;

A pop-up menu can be accessed by clicking on the icon corresponding to one of these elements. This menu allows the user to: define element's parameters; move, copy or delete the element; or edit the element's icon.

. Element

This menu allows the user to define the logic of the simulation model.

Routing - allows the definition of the Process Plans which describe the operations that must be performed on a part;

Arrival - schedules multiple arrivals (randomly or not spaced over time) of raw materials to the factory;

Schedule - schedules arrivals of raw materials to the factory which occurs only once;

Interrupt - schedules interruptions of model elements;

Resource - defines the resources which are items required by the Stations for processing the operations.

. **Display**

This menu allows the user to change the display parameters.

View - allows the user to pan across the factory and zoom in and out;

Paths - displays all the existing paths (i.e. Process Plans, Transporters and Conveyors);

Reports - allows the reviewing of the Reports from the last simulation run;

Traces - allows the reviewing of the Traces produced in the last simulation run;

Settings - allows the modifications of the background grid.

. **Simulate**

This menu allows the user to validate and run the simulation model.

Run parms - allows the setting of the parameters used during the running of the model;

Validate - validates a model before it is allowed to run;

Run - change to the Run-Time Environment;

The options described are used in the Layout Editor graphical environment to enter the factory layout information. The user interface is used [Kleine 1986] to simplify the data entry, to reduce data errors, and to help the modeller to manage and organize his data.

After the model has been described SIMFACTORY needs to do a consistency checking before running the simulation. This is made by selecting the option Simulate/Validate. If SIMFACTORY finds any errors it will report them to the user. After correcting the errors the user must repeat the process until no errors are found. When the model is error-free the user can start the simulation by selecting the Simulate/Run option.

The Simulate/Run option gives access to the Run-Time environment. In this environment the user is able to run the simulation and view new or old statistics and reports. The Run-Time environment has its own menu-bar which provide the following options:

Snapshots - allows the viewing of model statistics while the simulation is running;

Control - allows the changing of options about the simulation run (e.g. animation options, animation speed, Trace options and View options);

Review - allows the viewing of the Traces and Reports of the last completed simulation run;

Pause - suspends or ends the simulation run;

Help - allows the access to help information about the Run-Time environment;

Halt - ends the simulation.

While the simulation model is running SIMFACTORY provides a concurrent animated display. Using the Snapshot option the user can interrupt the simulation run, at any time, to see the model status. With this option the user is able to watch the status of processing stations, resources, queues, transporters, pending work orders, and future events. Furthermore, using the Snapshot option the user can also, control the animation and obtain logic traces.

During the simulation run SIMFACTORY generates periodic reports. Final or periodic reports can be obtained on [Kleine 1986] processing station utilization, raw material consumption, throughput, and resource utilization.

SIMFACTORY structure is much based on the performing of movement, processing and storage operations on objects called Parts or Workpieces. To implement these concepts SIMFACTORY uses the following modelling elements [Kleine 1986].

. Processing Stations

Processing Stations are elements where Parts are processed. They can be used to model different manufacturing operations. Several operations can be defined for a

station and also the same operation can be defined for different stations. Each operation has a setup time, an efficiency, and may requires the use of resources.

. **Queues**

Queues are elements used for storing parts or resources and are also used as entry points for raw materials.

. **Transporters**

Transporters are elements used to move parts and resources around within the factory. For each transporter it is necessary to specify its speed and the list of pickup and dropoff points;

. **Process Plans**

Process Plans are used to define how a subassembly is manufactured and consists of a set of one or more inputs, a sequence of operations, and one or more outputs.

. **Interruptions**

Interruptions are used to model maintenance, shifts, breaks, etc. on processing stations and transporters.

. **Resources**

Resources may be required by processing stations to perform operations and can be used to model jigs, fixtures, containers, operators etc.

. **Factory Layout**

The Factory Layout are used to define the processing stations, queues, and transporters to be included in the factory and how parts may flow through the factory.

. **The Product Description**

The Product Description defines which Process Plans should be included to represent the bill of materials.

. **The Production Schedule**

The Production Schedule consists of a sequence of work orders for final products. Each order specifies the final product, the quantity and the start time.

SIMFACTORY provides an easy and quick way of developing manufacturing systems simulation models. SIMFACTORY II.5 [CACI 1990], the SIMFACTORY latest version, provides a very user-friendly graphical environment. Within this environment the model can be built, using a menu-driven interface and manufacturing terminology, without any programming. Also, the use of icons to display the model running makes it very attractive to manufacturing people. A SIMFACTORY tutorial is given by [Kleine 1986] and a complete description can be found in [CACI 1990].

3.2.6. Simulation Support Environments

Mathewson [Mathewson 1989] defines a complete support system as one which aids all aspects of the use of simulation by providing individual modules for model building, control of experiments, results analysis and results presentation. As expressed in a recent panel [Kachitvichyanukul 1987] the Integrated Simulation Environment is the most talked about topic in the simulation software community in the past few years. The main objective of these systems is to provide the user with an environment that can give him support during all the stages of a simulation study.

Balci [Balci 1986] has characterised the life cycle of a simulation study in terms of 10 phases, 10 processes, and 13 credibility assessment stages. The life cycle diagram is presented in figure 3.13. The oval symbols represent the phases, the dashed arrows represent the processes linking the phases, and the solid arrows represent the credibility assessment stages. Looking to the diagram from figure 3.13 it is possible to identify only three main steps in a simulation study: problem definition, model development and decision support. However, the existence of many interactions makes a simulation study a complex task as shown in figure 3.13.

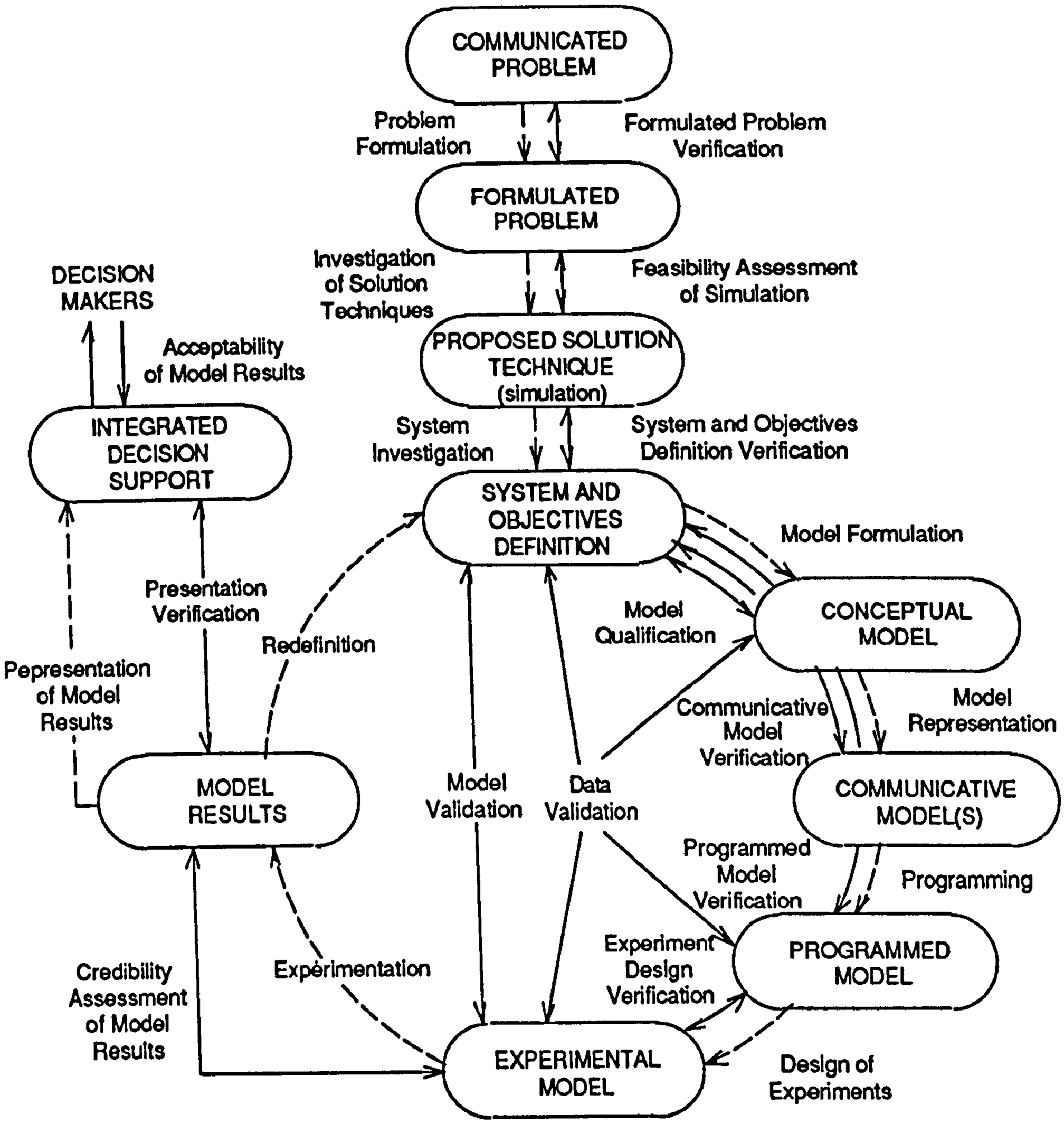


Figure 3.13 The Life Cycle of a Simulation Study [Balci 1986]

The size and complexity of simulations models has been growing fast in the last decade. The developments in computer hardware and software led users to be more demanding about user interfaces and supporting tools. Simulation has become a powerful method for modelling problems. It could be much more widely used if it were cheaper and easier. These are just some of the reasons that justify the need for simulation support environments. As expressed by [Nance & Arthur 1988] one of the distinctive characteristics of the human species is the search for a better way to accomplish assigned tasks. In this work they have explained the role of a methodology in the realization of a model development environment.

A review of related work was done by [Balci & Nance 1987] that have also presented a research prototype for a Simulation Model Development Environment (SMDE). This SMDE prototype has been developed to provide an integrated and comprehensive collection of computer-based tools to [Balci & Nance 1987]:

(1) offer cost-effective, integrate and automated support of model development throughout its entire life cycle;

(2) improve the model quality by effectively assisting in the quality assurance of the model;

(3) significantly increase the efficiency and productivity of the project team;

(4) and substantially decrease the model development time. The architecture of this SMDE is described in figure 3.14.

Several current research projects (see [Balci & Nance 1987]) shows the importance, the interest and commitment of the simulation community in this area. One of the good examples is the Computer Aided Simulation Model (CASM) project at the London School of Economics. In this project a group of researchers have been investigating ways of making the process of simulation modelling more efficient guided by a picture of an ideal environment. The CASM environment has as main components a problem formulator, an Interactive Simulation Program Generator (ISPG) and an output analyser.

An overview of the CASM environment is given in [Balmer & Paul 1986]. The CASM simulation modelling structure is based on a three-phase-based set of PASCAL simulation routines developed at the University of Lancaster [Crookes et al. 1986]. The use of Artificial Intelligence within the CASM environment is described by [Balmer 1987].

A rule-based expert system, a natural language understanding system and a fault diagnosis system were developed to support the earlier stages of model formulation [Doukidis & Paul 1985], [Paul & Doukidis 1986]. An ISPG called LANGEN [Chew 1986] was developed based on the extended Lancaster Simulation Environment (eLSE) and an improved version was later on called AUTOSIM [Paul & Chew 1987].

One of the early simulation support systems and also currently commercially available is The Extended Simulation Support System (TESS) [Standridge et al. 1987] of which a brief description is given next.

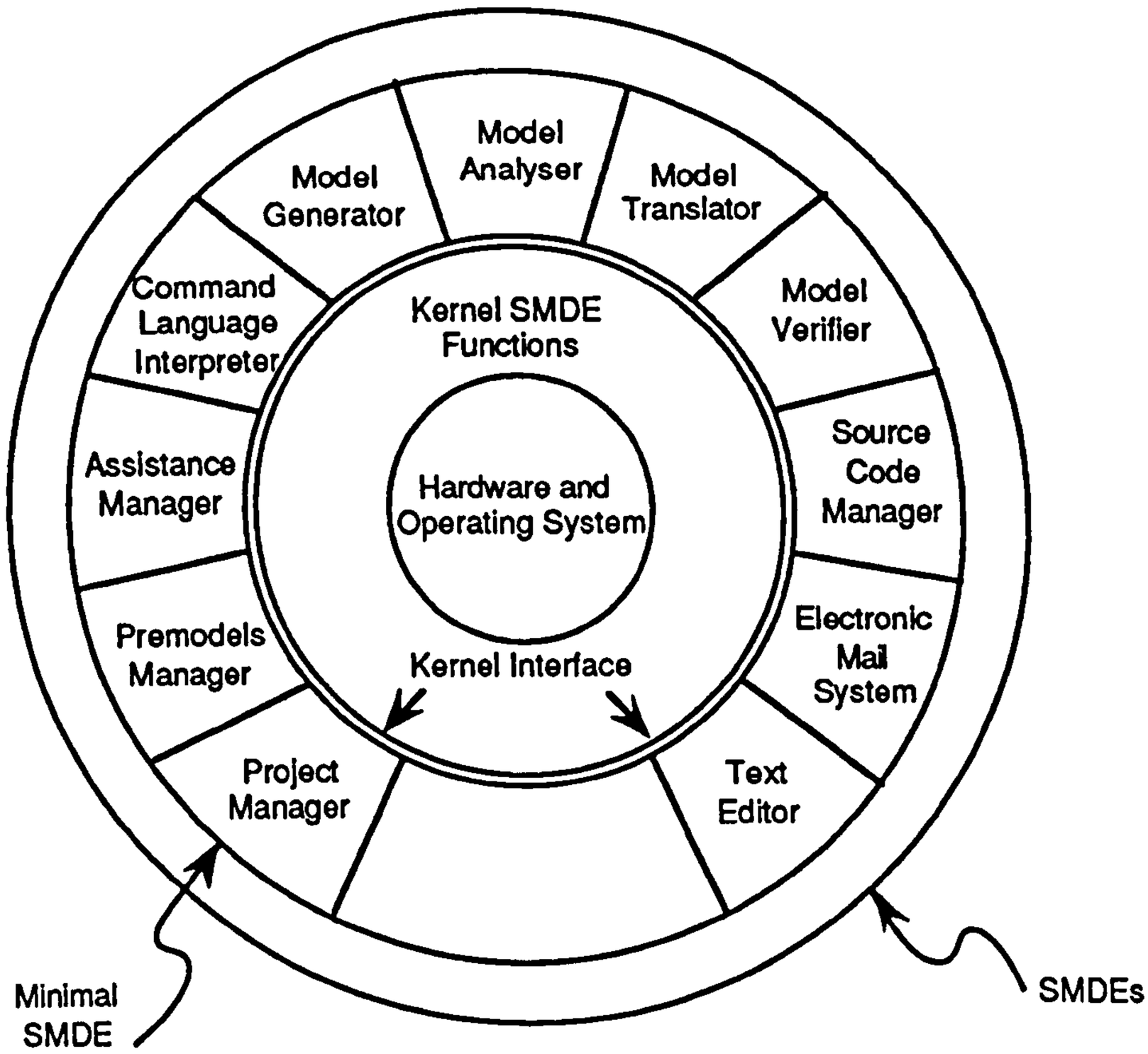


Figure 3.14 The architecture of the SMDE research prototype [Balci & Nance 1987]

3.2.6.1. The Extended Simulation Support System (TESS)

The Extended Simulation Support System (TESS) [Standridge et al. 1987] supports the model entry, simulation, statistical analysis, and result presentation tasks required in a simulation project. TESS can be used with SLAM II [O'Reilly & Lilegdon 1987], MAP/1 [Miner & Rolston 1986] or GPSS/H [Crain et al. 1987]. TESS provides a database where simulation data is stored. It can display variable values and statistics using business graphs and reports. TESS also allows animation to be used concurrently with simulations or after simulation runs.

Figure 3.15 shows the TESS problem solving framework [Standridge et al. 1986]. The top half of the oval represents the activities involved in problem solving using simulation. These activities are listed by [Standridge et al. 1986] as modelling, simulating models, analyzing and presenting results. TESS provides integrated features for the use of a simulation language, database manager, and graphics capabilities to

support these activities. The arrows in the diagram of figure 3.15 represent the links between the TESS features and the problem solving activities.

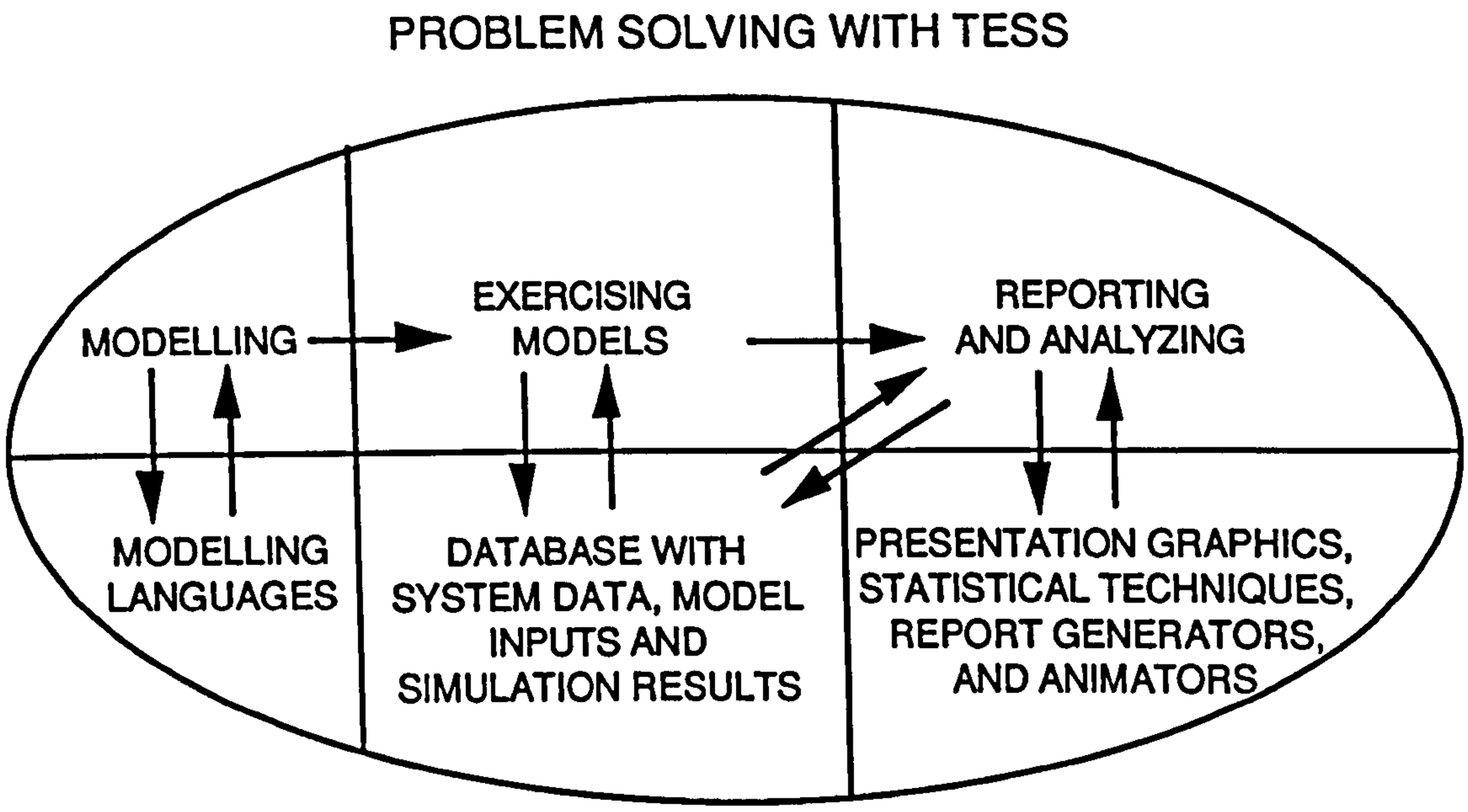


Figure 3.15 Conceptual View of TESS [Standridge et al. 1986]

According to [Standridge et al. 1986] TESS has the following features:

- a framework for problem solving using simulation;
- separation of the analysis and presentation of simulation results from their generation in simulation runs;
- integration of modelling and simulation execution with reporting, graphing analysis and animation capabilities;
- a command language to access each capability used in problem solving;
- creation and management of SLAM II network models;
- independent specification of experimental conditions for controlling simulation runs (CONTROLS);
- management of user defined simulation input data;
- a report generator for presenting simulation results and other data;
- graphing of networks, simulation results and user defined data;
- procedures for dynamically presenting the operation of a model, that is, the animation of simulation results both after and concurrently with simulation;
- support for database management tasks.

The diagram of figure 3.15 describes how TESS is used to support the different stages of a simulation project. Tutorial descriptions of TESS can be found in [Standridge et al. 1986] or [Standridge et al. 1987].

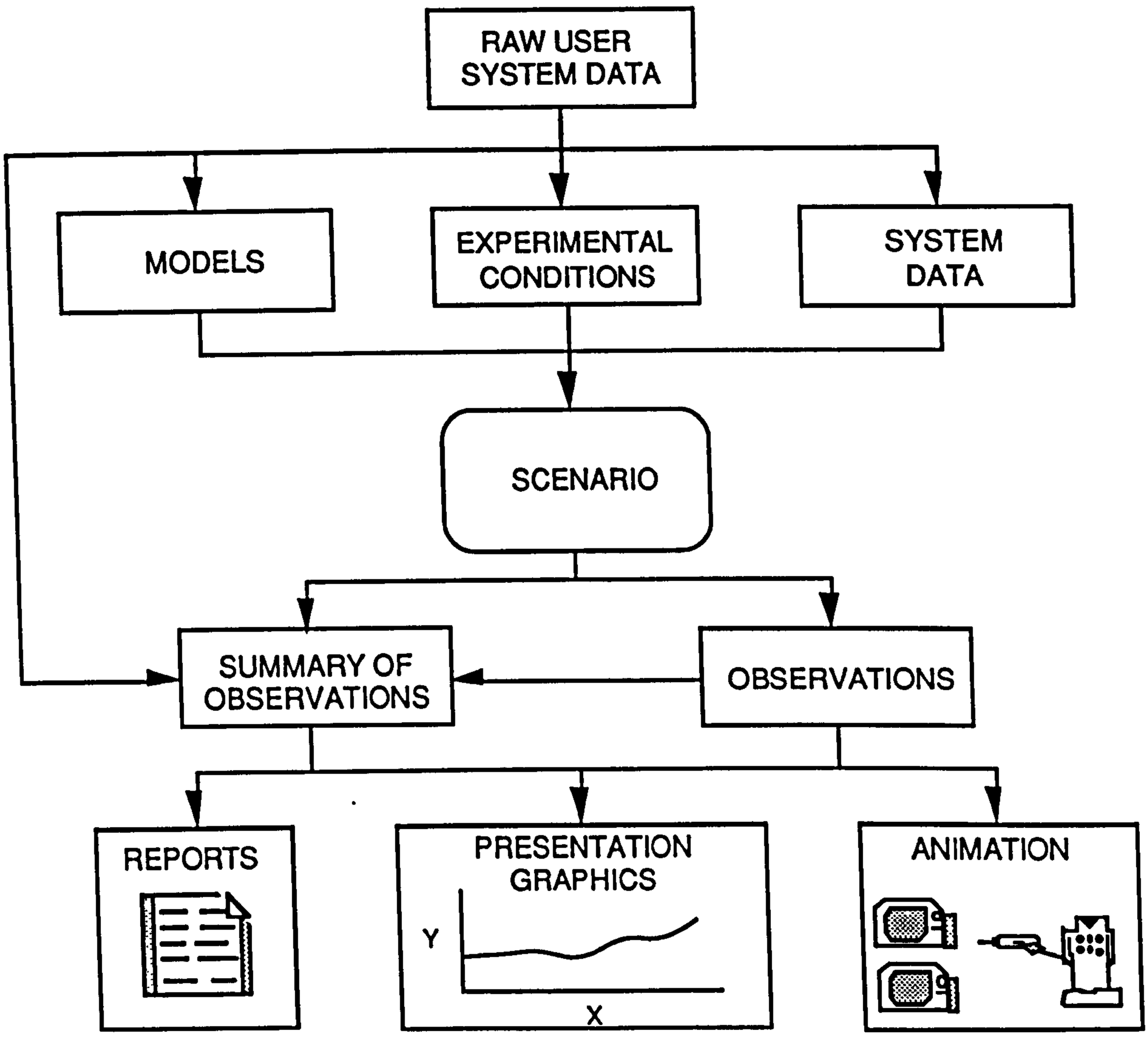


Figure 3.16 TESS Simulation Project Framework [Standridge et al. 1987]

3.3. Summary

When a simulation model has to be developed a large number of options are available. In each case the choice of the more adequate simulation system depends on several factors such as: the users simulation and programming expertise, the characteristics of the system to be simulated, the project's budget and timescale.

The users expertise is the key factor in finding feasible solutions to develop a simulation model. This expertise dictates the kind of simulation system that can be used to develop a model meeting the project's timescale.

Simulation systems can be grouped under the following categories according to their model building approach: simulation libraries, simulation languages, graphical block languages, program generators, data-driven generic models and support environments. The description given and the examples of this chapter shows the type of skills required to develop a simulation model using a simulation system within each one of these categories.

As the number of simulation studies increases it becomes more difficult to find analysts with the required level of expertise. So potential simulation users tend to have lower expertise in programming, simulation and modelling. These users need simulation systems easier to use and that can give them support during all the stages of the simulation project. As shown in this chapter the simulation systems using the data-driven generic models and support environment approaches are the most appropriate to achieve these objectives.

CHAPTER 4

THE NATURE OF WAREHOUSE SIMULATION AND THE APPROPRIATENESS OF EXISTING TECHNIQUES

4. The Nature of Warehouse Simulation and the Appropriateness of Existing Techniques.

4.1. Introduction to Warehouse Design

Most of the principles associated with modern warehouse systems and materials handling equipment have been known for more than one century. Since 1950 these principles have been applied to make warehouses and materials handling more efficient. However, it is only, since the late sixties that their cost and implications in the distribution chain have become more clearly understood. Although, the word 'warehouse' means just 'a building in which goods, or wares, are stored' the main functions of a modern warehouse are the following [Firth 1983]:

- "To provide an adequate buffer storage against inequalities caused by unpredictable variations in supply and demand. This may be to guard against loss of production, to ensure batches of economic size, to provide marketing back-up by maintaining availability of spares and adequate service levels, and on occasions to provide a speculative hedge against anticipated inflation."

- "To safeguard stock from damage, deterioration and unauthorised removal by providing an environment which is appropriate to the materials being stored."

- "To record accurately receipts, stockholding and despatches, and to provide an efficient communicational interface with all appropriate parts of the system being served."

The design of warehouse systems involves the determination of throughput and space requirements and their transformation into integrated systems of people, equipment, and space [Bozer et al. 1982]. Bozer et al. describe modelling concepts used in warehouse design with special emphasis on data base development and analysis, analytic models, and simulation modelling as a design/evaluation tool. Bozer et al. and later Branigan [Branigan 1988] have described the warehouse design process as having the following phases:

- structuring (problem definition);
- data collection and analysis;

- generation of design options;
- evaluation of options;
- selection of the preferred option;
- rigorous testing.

In the structuring (problem definition) phase the objectives and scope of the project are established. During this stage strategic plans should be outlined considering materials handling, warehouse locations, safety, image, design and cost goals etc.

The data collection and analysis phase is essential to provide an accurate image of the distribution system, from which the warehouse requirements can be specified. The importance of this phase is well expressed by the words of Egarr's colleague [Egarr 1984] about warehouse design 'Give me a few facts and I'll design a warehouse. Give me all the facts and I'll get it right!'. The data to be collected covers distribution operations, warehousing/corporate systems and product characteristics, activity and volume throughput, and customer service. This data may be obtained from existing sources or estimated as it will be the case of a totally new business. The analysis of the collected data provides the necessary information for the definition of the design parameters (e.g. unit load size, storage and handling requirements etc.).

Once all the warehouse requirements have been defined it is possible to generate the design options. The number of solutions available is usually very large and the search for feasible options can be an iterative process that is time consuming and requires experience in distribution and warehousing. The result of this phase is a set of options specifying information such as the warehouse layout, building sizes, storage requirements, material handling equipment, labour as well as cost estimates.

The evaluation phase involves the detailing of each option in order to determine the lowest cost solution that meets the business requirements. Simulation modelling techniques and analytic models [Bozer et al. 1982] can be used to refine and assess each warehouse system. Although, some level of testing can be made to determine if each solution will work after implementation the final decision is mainly based on the designer's experience.

The selection of the preferred option is made according to the chosen criteria. Many factors can influence the final decision (e.g. cost, flexibility, image, etc.) but again the designer's experience is vital in selecting the better solution. The objective is to select

the warehouse system which provides the most efficient and economic flow of goods at a minimum cost of space, labour and equipment. Before the implementation of the warehouse system, a detailed description must be produced and also rigorous testing must be done.

4.2. A Decision Support System Framework for Warehouse Design

The description given in 4.1 shows that warehouse design is a complex task for which success relies mainly on the designer's experience. The large number of technological options available and the difficulty in evaluating them justifies the search for better and more effective tools for warehouse design. The trend, in recent years, of building warehousing systems larger and automated has made warehouse design become even more complex.

The nature of warehouse design requires the manipulation of large amounts of data and is often an iterative process that forces the designer to go through the different design phases (see 4.1) several times before reaching the final solution. This suggest an integrated computer environment that can give support to the user during all the design phases. With this objectives in mind a Decision Support System (DSS) framework for warehouse design is proposed in figure 4.1.

This system is intended to help the less experienced designer and to serve as a supporting tool in defining, generating, evaluating, selecting and testing different options during a warehouse design project. The DSS has two main sections: the Outline Design; and the Detail Design. The Outline Design corresponds to the first three design phases (see 4.1): structuring; data collection; and generation of the design options. The Detail Design corresponds to the last three design phases (see 4.1): evaluation of options; selection of the preferred option; and rigorous testing.

In the Outline Design section the following functions would be performed.

. General Parameters

This block corresponds to the first two design phases (see 4.1): structuring; and data collection and analysis. During this stage all the necessary data to characterise the problem would be defined. The data would be kept in a Data Base allowing

easy access for modification and later use. An expert front end could be used to guide the less experienced warehouse designer through the data definition process. The access to a library of projects would allow the use of data from similar situations in a new project. A library of statistical based functions would enable data analysis to be performed and also, the generation of data when real data was not available (e.g. in a green-field situation).

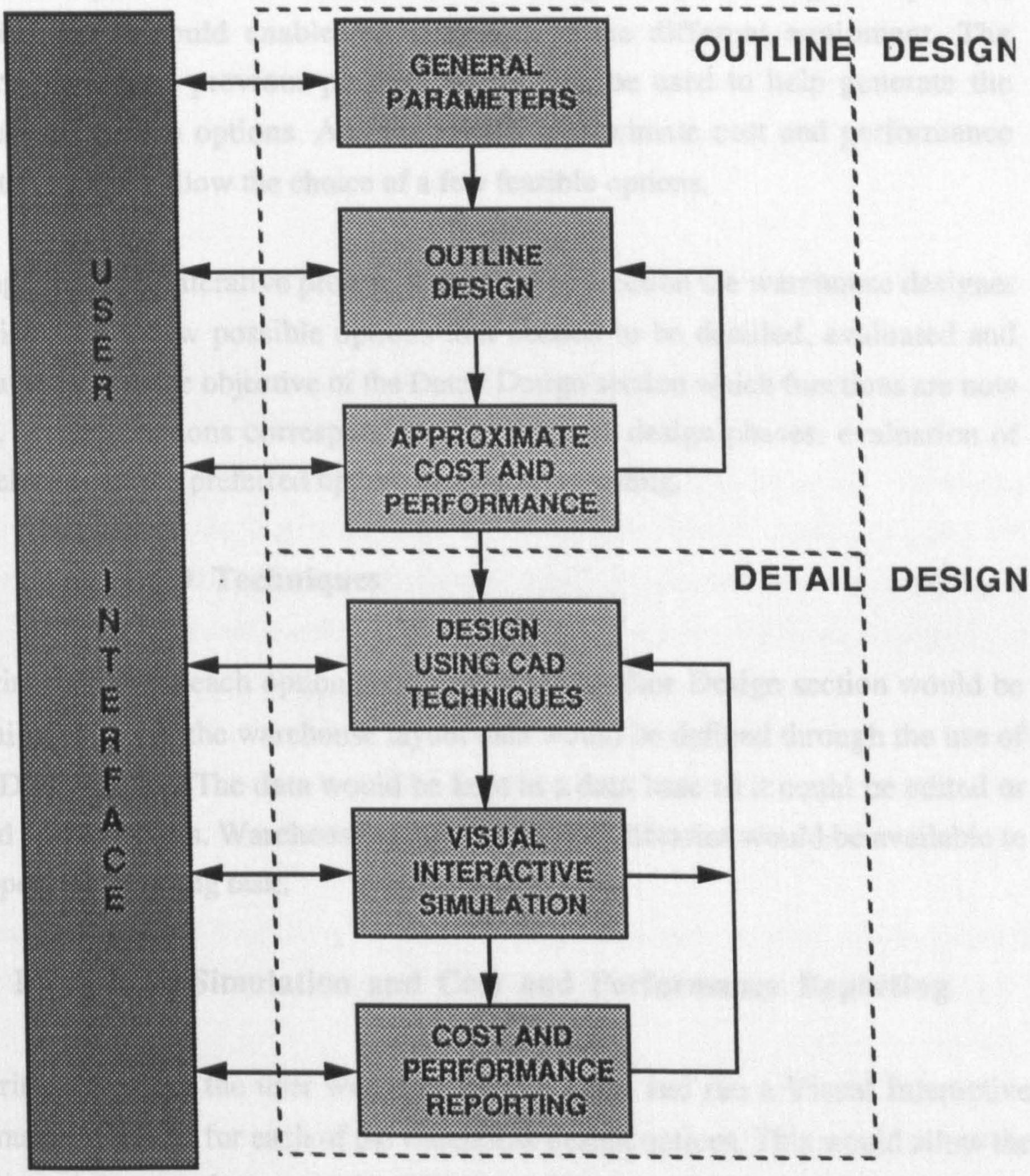


Figure 4.1 A Decision Support System Framework for Warehouse Design

. **Outline Design and Approximate Cost and Performance**

These blocks correspond to the third design phase (see 4.1): generation of the design options. During this stage a set of warehouse design options would be generated. An expert front end could be used to give support to the less experienced warehouse designer in the search for better solutions. The design process would be based in an interactive dialogue. This dialogue and the data defined in the General Parameters stage would allow the definition of the warehouse parameters. A data base of technological options (e.g. transporters, racking, etc.) would enable the selection of the different equipment. The information from previous projects would also be used to help generate the warehouse design options. A module with approximate cost and performance functions would allow the choice of a few feasible options.

After going through an iterative process in the Outline section the warehouse designer would finish with a few possible options that needed to be detailed, evaluated and tested. That would be the objective of the Detail Design section which functions are now described. These functions correspond to the last three design phases: evaluation of options; selection of the preferred option; and rigorous testing.

. **Design Using CAD Techniques**

During this stage each option generated in the Outline Design section would be detailed. Most of the warehouse layout data would be defined through the use of CAD techniques. The data would be kept in a data base so it could be edited or used at later stages. Warehouse equipment symbol libraries would be available to support the drawing task.

. **Visual Interactive Simulation and Cost and Performance Reporting**

During this stage the user would be able to build and run a Visual Interactive Simulation Model for each of the warehouse design options. This would allow the user to evaluate and compare the different options as well as test their performance according to the warehouse design requirements. A set of cost and performance reports would help the user to select the preferred option. In the case of an unsatisfactory solution the warehouse design data could be edited and the process

repeated again. An expert front end would help the user to analyse the simulation results.

The system described in figure 4.1 would represent an invaluable tool for warehouse design. The first section, the Outline Design, would be much more useful for the less experienced warehouse designer as it would give him support in the definition of the warehouse overall parameters. The last section, the Detail Design, would be the most important because it would allow the warehouse designer to evaluate the different options and make sure that the selected one would work according to the specifications. Simulation is probably one of the best methods available to the warehouse designer which allow him to analyse the complex relationships between all the components of a modern warehouse.

4.3. The Need of Warehouse Simulation

From the description of the Decision Support System framework for warehouse design (see 4.2) it is clear that simulation has a very important role in the warehouse design process. Some years ago the use of simulation was very expensive, due mainly to computer costs; it also required a high level of expertise. Today, computer costs are affordable and the developments in software, specially in the user-interface area, have made simulation a more acceptable technique. Stenzel [Stenzel 1988] has analysed the application of simulation within the material flow technique and logistics.

The use of simulation for warehouse design has only recently become common. The need of warehouse simulation has been growing in the past few years mainly because:

- warehouses are becoming larger involving budgets of several million pounds which means that any design improvements can conduct to important savings;
- investments in warehouse systems are so high that the design must be right;
- a poor design of a warehouse can have serious implications on the whole distribution system;
- as the level of warehouse automation is increasing, warehouses become more inflexible;

- warehouse systems complexity make it very difficult to evaluate different alternatives;

- the use of computers for warehouse management, the rigid logic of mechanised system and the standardisation of handling units make warehouse systems much easier to simulate.

All these reasons justify the use of simulation in warehouse design. However, the question is to know which of the available solutions to use and whether they can be used effectively.

4.4. Available Solutions for Warehouse Simulation.

A large number of simulation systems are available today, each one having its own strengths and weakness. In Chapter 3 an overview of the current approaches to model building has been given. The simulation systems were classified using a criteria based on how the user had to develop the model and what kind of skills were required. This perspective was used to support the idea that who ever has a problem to be solved using simulation should be directly involved in the development and use of the model.

Three important factors in the choice of a simulation system are: the experience of the person who will be involved in the development of the model; the characteristics of the system to be simulated; and the estimated time to develop the simulation model.

The experience of the user is fundamental in the choice of the simulation system to use. A user without programming or simulation experience should consider first the use of a Data-driven Generic Model (see 3.2.5.), a Simulation Support Environment (see 3.2.6.) or a Program Generator (see 3.2.4.). However, the choice of one of these approaches would be dependent on finding a simulation system that can handle the problem with the required level of accuracy. The simulation systems based on these approaches are usually easy to use, have short learning periods and allow a fast development of simulation models.

When the user is unable to find a simulation system capable of solving his problem, within one of the above approaches, then he must look into the more general

approaches: Simulation Libraries (see 3.2.1.); Simulation Languages (see 3.2.2.); or Graphical Block Diagram Languages (see 3.2.3.).

If the user is already familiar with a programming language he may try to use a library of simulation subroutines such as SEE-WHY (see 3.2.1.2.). This option would take advantage of the user's previous programming experience and would allow him to concentrate on learning the simulation concepts and how to use the simulation subroutines. Simulation libraries are available for different programming languages (e.g. FORTRAN, PASCAL, C). However, difficulties may arise if the simulation library does not have enough functionality. In this case, either the user has access to the simulation subroutines source code and is able to develop them further, or he will be forced to look for another option.

If the user is not familiar with any programming language then he should look into a simulation system within the Simulation Languages or Graphical Block Diagram language approaches. The choice should be made considering the ease of use, the facilities provided and the appropriateness of the simulation system to solve the kind of problem the user has.

The graphical block diagram languages offer some advantages to the less experienced user such as simple modelling concepts and the facility of building the model using graphic block diagrams (see 3.2.3.). The use of graphic blocks although being a straightforward way of building simulation models can become impracticable for very complex models. However, they always are useful during the simulation language learning stage, for building small models or even in the definition of parts of more complex models.

The use of a simulation system within one of the Simulation Libraries, Simulation Languages or Graphical Block Diagram Language approaches, requires a high level of expertise and often long development periods. The time required for developing a simulation model depends not only on the user's experience but also on the ease of use of the simulation system. Usually an easy to use simulation system is associated with less development time.

The diagram from figure 4.2 shows a qualitative comparison between the different simulation approaches described in Chapter 3. This diagram shows that the advantages in ease of use and less development times are achieved by reducing the level of

generality. The Data-driven Generic Models approach being easy to use, requiring less development time and user expertise stands as the best solution. Unfortunately, the simulation systems based on this approach are dedicated to particular areas having so a limited use.

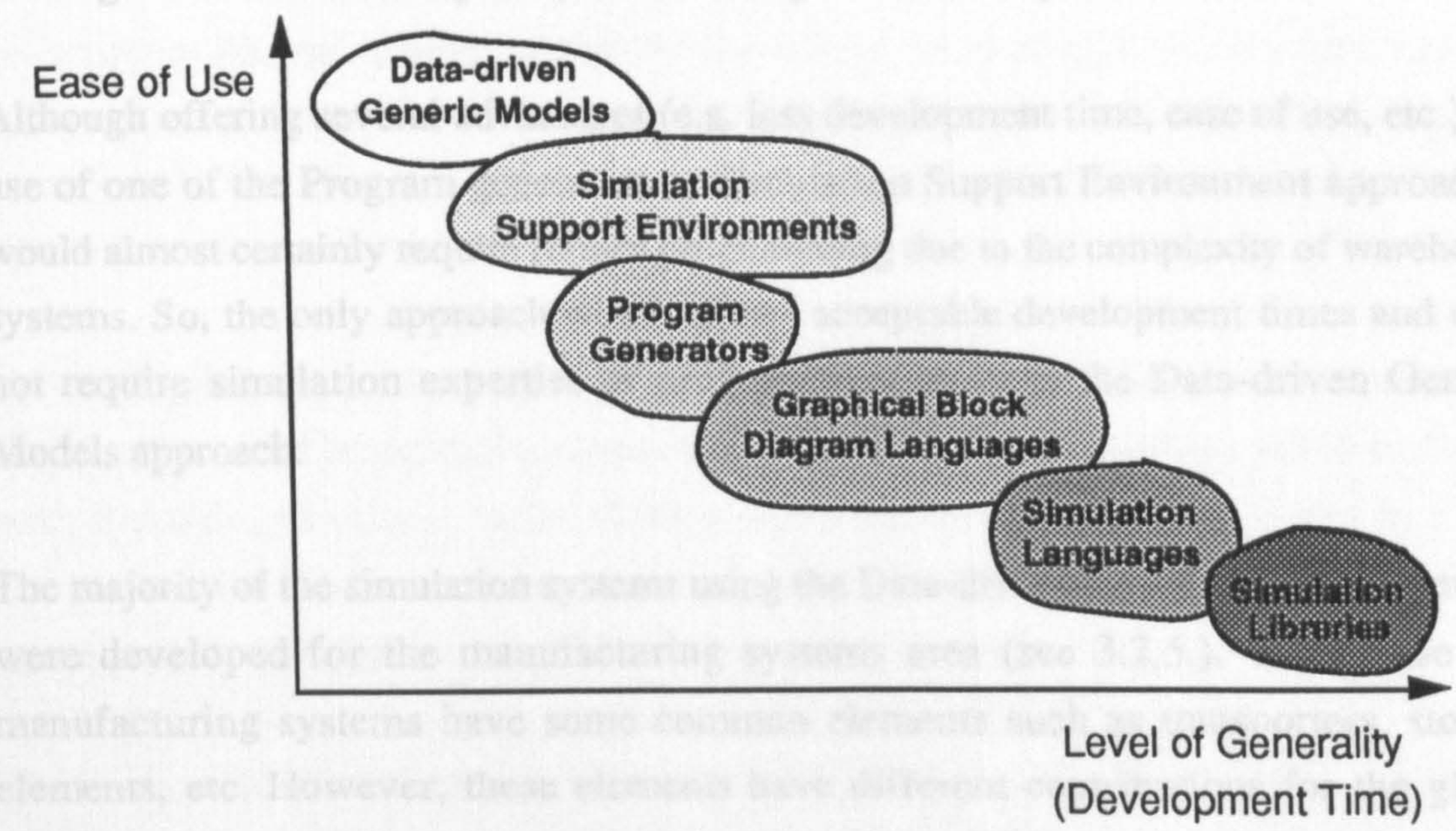


Figure 4.2 A Qualitative Comparison Between the Different Simulation Approaches

What have been said can be appropriate for different areas as well as for warehouse simulation. However, warehouse systems have certain characteristics that often make the building of simulation models a very complex task. Among these characteristics are the large volumes of information, the complex relationship between the warehouse elements, the different control rules and the need for a high flexibility in changing model parameters.

When the need for a warehouse simulation model is identified the development can be made within the warehouse design team or by an external consultant. Although, the use of a consultant can speed up the process of obtaining results and gives professional expertise it results in a high cost with very low flexibility. Furthermore, the increasing use of simulation means that it will not be always easy to find an available consultant. These reasons, together with the fact that there are many advantages in involving directly the warehouse designers in the model development, make it is desirable that the warehouse simulation should be implemented by the design team.

In a warehouse design team it not very common to find someone with simulation modelling expertise or with a reasonable programming experience. This means that the choice of the Simulation Library, Simulation Language or Graphical Block Diagram Language approach would lead to an unacceptable development time due to the long learning curve and the complexity of modelling warehouse systems.

Although offering several advantages (e.g. less development time, ease of use, etc.) the use of one of the Program generators or Simulation Support Environment approaches would almost certainly require further programming due to the complexity of warehouse systems. So, the only approach which allows acceptable development times and does not require simulation expertise or programming skills is the Data-driven Generic Models approach.

The majority of the simulation systems using the Data-driven Generic Models approach were developed for the manufacturing systems area (see 3.2.5.). Warehouse and manufacturing systems have some common elements such as transporters, storage elements, etc. However, these elements have different contributions for the global efficiency of manufacturing and warehousing systems. While in manufacturing they are often treated as auxiliary elements, in warehousing they are key factors in the overall performance of the system. Movement is the prime activity in a warehouse; machining processing is the prime activity in a production unit.

When simulating manufacturing systems it is common to use average values for transporter travel times and define their path as a series of pickup and dropoff points. Storage elements are modelled as buffers having a certain capacity and a queue discipline. The number of buffers in a manufacturing system is usually small and, although they can have a defined physical location in the system, each element in the buffer is not physically identifiable. This level of detail is good enough for most of the manufacturing systems where the key factors are the flow of parts through the system, the process plans and the utilisation of the processing machines.

Data-driven Generic Model simulation systems such as WITNESS (see 3.2.5.1.) or SIMFACTORY (see 3.2.5.2.) can be used to simulate warehouse systems. The pallets, racking storage and transporters can be simulated using the Parts, Buffers and Vehicles of WITNESS or the Parts, Buffers and Transporters of SIMFACTORY. Both WITNESS and SIMFACTORY provide user interfaces which allow any warehouse

designer to develop a simulation model in a reasonable period of time even without previous simulation experience.

When using a simulation system like WITNESS or SIMFACTORY to develop a warehouse model some simplifications need to be made. The locations of the racking cells cannot be realistically modelled. The number of racking cells in a warehouse is typically in the order of several thousands which makes unfeasible the individualization of each cell. One possible solution is to define each racking aisle as a Buffer and the access time as an average value.

Some simplifications have also to be made concerning the modelling of the transporters. Their path cannot be defined accurately and situations like congestion has to be handle with approximate values. As the racking layout and the transporters are key factors in the performance of a warehouse system these simplifications may compromise the success of the simulation study.

The warehouse simulation models built with WITNESS or SIMFACTORY can be used for a first approach in the evaluation of a warehouse system or to model only certain areas within a warehouse. They are not capable of handling all the complexity of a warehouse system and cannot represent with an acceptable level of accuracy the relationship between all the warehouse elements. These models would be often insufficient in the detailed evaluation of different options, and in the rigorous testing of the selected option.

The approach used by WITNESS, SIMFACTORY or similar simulation systems enables experts in certain areas (e.g. manufacturing, warehousing, etc.) to develop simulation models and use them effectively as a helping tool for finding better solutions to their problems. This is only possible by dedicating the simulation system to a particular area (e.g. manufacturing systems) which makes it inadequate for use in other areas even if they share some similarities (e.g. with warehouse systems).

Warehouse systems have special characteristics (see 4.6.) that require a new way of defining the simulation model and new modelling elements to represent the complex logic of a warehouse system. There follows some examples, showing the traditional approaches to warehouse simulation.

4.5. Some Examples of Warehouse Simulation Models

In the past few years simulation has been used frequently to analyse warehouse systems. Most of the simulation models have been developed using general purpose programming languages or simulation languages which provide the necessary flexibility to model systems as complex as warehouses. In many cases it would be too complicated to simulate the whole warehouse system, hence only the most critical areas were simulated.

Pidd [Pidd 1986] described a simulation project developed for the parts operation of a large UK motor manufacturer. The simulation model was developed using BASIC on an Apple II microcomputer. An icon-based graphics screen allowed the display of the entities and colours were used to show their various status during the simulation. The user was able to define some parameters during the initialisation phase such as arrival rates of vehicles, speed and number of trucks, space in each bay, processing times, etc. An optional final print out was available to the user with the model final conditions and some summary statistics. According to Pidd [Pidd 1986] the model has enabled the designers to experiment with various options for the replanning of the goods inwards area of the warehouse.

An interactive simulation modelling of Automated Storage Retrieval Systems (AS/AR) was presented by [Raghunath et al 1986]. This package allowed the user through an interactive menu to select a combination of modules that defined the AS/AR system. Five modules were available: a Central Module where it defined the main part of the AS/AR (e.g. number of aisles, dimensions, number of slots, crane characteristics, etc.); a Transporter Module where it was defined the transportation methods to interface with the AS/AR (e.g. AGVs, lift trucks, conveyors or monorails); a Material Processing Function Module where it was defined the functions performed on materials before and after been storage; a Storage Configuration Module; and an Input/Output Configuration Module.

For each module the user had to specify the values of the associated system variables. The defined data was used to generate the simulation model source code in the SIMAN language (see 3.2.3.2.) which could then be executed. The AS/AR system performance statistics were available in a final SIMAN output report. No graphics facilities were available although it was mentioned by the authors as one of the future enhancements. The package was developed for running on the IBM-PC microcomputer.

The SIMAN language was used again in an AS/AR simulation study [Pulat & Pulat]. The authors have presented a combinatorial approach to evaluating the throughput performance of an AS/AR, with simulation as the primary method of investigation. The SIMAN language offers a good development base for these type of problems as it provides special purpose constructs oriented to the modelling of handling systems (see 3.2.3.2.).

McGinley [McGinley 1988] has described the use of Visual Interactive Simulation (VIS) (see 2.3.) in a consultancy project as an integral part of the design of a new warehouse facility for the Mobil Oil Company. The warehouse was to be a single automated highbay served by a sophisticated Automated Guided Vehicle (AGV) transport system. As the main interest was in the capacity and control of transport systems some simplifications have been made due to the large number of product types and sizes. The model was developed using SEE-WHY (see 3.2.1.2.) from ISTEL. Two mimic diagrams showed the whole track and the major system elements. During the simulation run the AGV's were seen to move round the track and colour coding was used to represent their current state. The simulation could be stopped at any time to inspect and/or change any of the key variables.

Keith and Porter [Keith & Porter 1988] have presented a computer simulation project carried out as part of the design phase of a new distribution centre for Baxter UK Limited, a medical supplies company. The centre combined automated and manual systems. It had an Intake area where all the goods were received. The goods were then taken to a Storage Cell by a conveyor system. Emergency medical orders were handled by keeping a representative stock of every product in a manually accessible location called Forward Picking. The consolidated orders corresponding to a shift requirements were issued in the Load Accumulation area and then manually sorted to order pallets.

The simulation model was developed using OPTIK (see 2.3.1.). The manually operated functions were represented in detail in the graphics display, whereas the automated pallet cranes were just displayed in summary. The simulation model has covered only the most critical areas in the distribution centre. The experimentation carried out has enabled the evaluation of different equipment levels, alternative equipment combinations, equipment breakdown, various priority changes and product range changes.

Rietz [Rietz 1985] has presented a warehouse simulation model developed with SEE-WHY for Multipart - Leyland Vehicles spare parts organisation. The model allowed assessing the trade-off between additional or faster cranes or AGV's against better control rules, to achieve planned throughput. Rietz has argued that a simulation model of a warehouse is the only effective means of ensuring that it is right-first time.

Simulation has also been used associated with other techniques to help solve problems in warehouses. Examples of this are the optimisation of a steel plate warehouse [Minnee 1988] and the use of real time artificial intelligence and simulation in warehousing [Hitchens 1987].

Warehouse simulation models are usually very complex requiring flexible and powerful simulation tools and long development times associated with high costs. Rietz has estimated for an average simulation work, similar to the one described in [Rietz 1985], a development time of three months and a cost of £20000 (1985 based cost). The cost of a simulation study is not significant when compared with the cost involved with an warehouse project and the results obtained by the use of simulation can largely compensate the investment. However, large simulation development times and the lack of flexibility caused by building dedicated models can compromise the effectiveness of the use of simulation.

There are certain situations where warehouse simulation can be handled with a Data-driven Generic Model simulation system such as WITNESS (see 3.2.5.1.) or SIMFACTORY (see 3.2.5.2.). These situations are not very common and often correspond to large simplifications of the warehouse system or concentrate their attention only on critical areas of the warehouse.

In the majority of warehouse simulation studies it is inevitable the use of a simulation system requires high levels of expertise and programming skills. The advantages of Visual Interactive Simulation (VIS) (see 2.3.3.) makes it almost indispensable in current warehouse simulation models. Animation can help to understand, reveal problems and stimulates creativity when analysing the complex relationship between different warehouse components.

Two case studies are described next. The first one demonstrates that an easy to use simulation package can have limitations which make it inadequate for simulating even simple warehouse systems. The second one reveals the amount of effort and expertise

4.5.2. A Warehouse Simulation Model Using SIMVIS

A simulation model of a full automated warehouse was developed for EFACEC a large Portuguese company in the warehouse design, building and consultancy business. EFACEC was preparing a proposal for a client and they needed the model to evaluate and test their options. They also wanted to use the model as a marketing tool to gain the contract.

4.5.2.1. Warehouse design options

EFACEC's engineers had two design options that were identical except in the number of aisles and in the number of racking levels. The first option has six aisles with five levels of racking, and the second one has five aisles with seven levels of racking. In figure 4.4 the warehouse layout of the six aisle option is presented. The warehouse must be capable of storing all the products coming from the different production plants and capable of supplying the company's retailers.

The warehouse (see fig. 4.4) has two bays at the reception and can unload simultaneously two trucks. The products arrive by truck in full pallets with 36 cases each. The pallets are unloaded automatically by roll conveyors. Then are transferred to a chain conveyor by a lift table.

One by one each pallet is moved by a rail trolley to another chain conveyor. During this path the pallet is identified by a bar code reader and has its overall dimensions automatically inspected. If any problem is detected the pallet is directed to a different circuit (turn left in the end of the trolley path) for manual inspection. When the pallet is in good condition it is taken by the chain conveyor (on the right at the end of the trolley path) to another rail trolley.

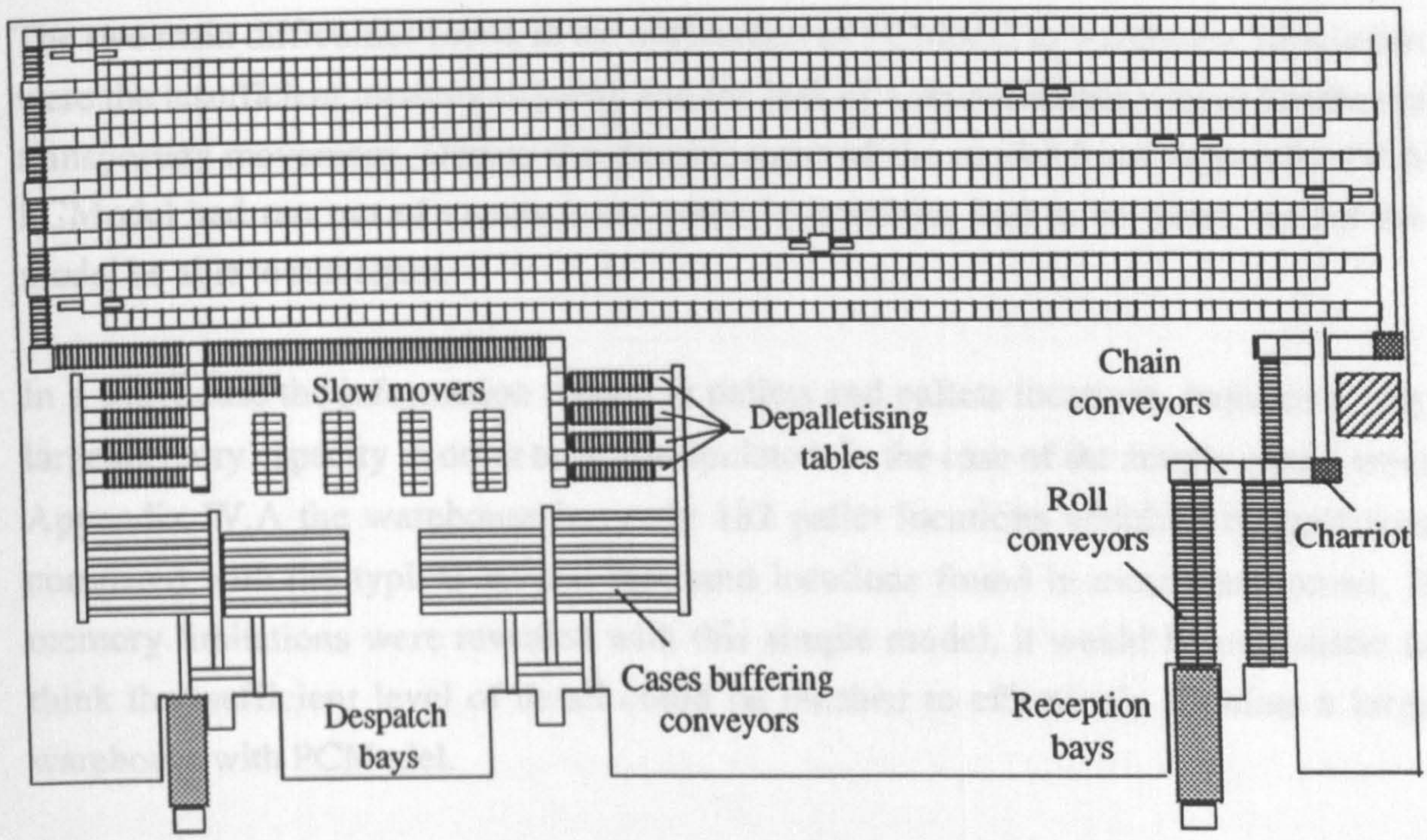


Figure 4.4 Fully automated warehouse layout

This trolley moves the pallet to a chain conveyor on one of the racking aisles, according to the operational conditions. The pallet then waits in a table at the conveyor's end until it is picked up by the crane and put away in the racking. Each racking aisle has a dedicated crane. The crane operational characteristics are listed in table 4.1.

Horizontal speed	Vertical speed	Forks speed
1.6 m / s	0.6 m / s	0.5 m / s

Table 4.1 Racking cranes operational characteristics

The warehouse has two identical despatch docks (see fig. 4.4), each one with two bays. Two trucks can be at the same time in one dock but only one can be loading. The existence of the two bays allows the preparation of one truck while the other one is loading. The product cases are loaded into the truck by conveyors. The loading sequence is defined by the computer according to the order information. As the truck must visit different company's retailers throughout the country the loading sequence must be made in the reverse order of the retailer's visit sequence to allow easy unloading.

The two main difficulties found in the application of PCModel to warehouse simulation were the insufficient memory capacity and the lack of a more flexible way to handle the transporters movement. During the development of the model from Appendix IV.A PCModel had run out of memory and some instructions had to be taken out for the model be able to run again.

In a warehouse the information related to pallets and pallets locations, requires a very large memory capacity in order to be manipulated. In the case of the simple model from Appendix IV.A the warehouse has only 132 pallet locations which is insignificant compared with the typical several thousand locations found in most warehouses. If memory limitations were revealed with this simple model, it would be unrealistic to think that sufficient level of detail could be reached to effectively simulate a large warehouse with PCModel.

PCModel handles the movement of objects through the use of special move instructions which sequence define the object path. These instructions together with decision making logic are grouped in blocks called routes. PCModel provides instructions to do absolutes moves (MA) to a certain column-row coordinate or relative moves down (MD), to the left (ML), to the right (MR) or up (MU). These instructions and the routes definition are well adapted to most manufacturing systems where the number of possible paths is small. Usually a different route block is created for each different path.

In warehousing as the transporters must access to many different locations it would be impracticable to define all possible paths one by one. Even in the case of the model from Appendix IV.A the number of different path combinations are given, for each group of transporters, by three (bays) times 132 (cells) equal to 396.

The solution adopted in the case of the model from Appendix IV.A was the development of a subprogram, called Link in PCModel, that calculates the path between two points in the warehouse and move the transporter along that path (see final part of the listing from Appendix IV.A). Although, this subprogram was able to compute the different transporter paths it is dedicated to this particular situation. So, in a different situation another algorithm had to be developed. In the case of a complex network it could be difficult and time consuming to find a solution. It is clear that the way PCModel handles the movement of objects does not give enough flexibility to define the complex movement of transporters within a warehouse.

represented by a character box, when in an empty state, or by the letter corresponding to the pallet they are moving. The movement of transporters is made along the right side of the aisle and congestion is realistically simulated as PCModel does not allow more then one object at the same overlay location.

In appendix IV.A a listing of the warehouse model language statements are presented. Most of the model parameters can be easily changed using the PCModel text editor. The following parameters are entered interactively in the beginning of the simulation: warehouse fill percentage; Outorders inter-arrival time; average number of pallets per Outorder; SD of pallets per Outorder. Before starting the simulation the warehouse is filled up to the warehouse fill percentage and the number of pallets of each product group are defined according to the product group throughput percentage. The Outorders inter-arrival time is used to generate the Outorders arrival time using a negative exponential distribution. The average number of pallets and the SD are used to define the number of pallets in each Outorder using a normal distribution.

The Inorders are generated automatically when the number of product group pallets reaches a minimum level. The minimum level, the number of pallets in each Inorder and the time delay until the Inorder arrival are defined as parameters (see Appendix IV.A). Other parameters defined are the time delay to pick a pallet from a cell and the time delay to pick a pallet from the reception. Data is collected to produce statistical reports such as the transporters utilisation (%) per hour. The statistical report screen can be displayed at any time by pressing a PCModel command key.

The development of this model had the objective of evaluating the appropriateness of the PCModel language for simulating warehouse systems and to assess the amount of effort involved and the kind of expertise required. The model although not developed to simulate a real system provides sufficient functionality to enable the drawing of conclusions.

PCModel provides an easy to use simulation environment and an intuitive way of defining the simulation model. However, it still requires large programming experience especially if the simulation model is complex (see Appendix IV.A). As a consequence of this the learning curve and model development time can be significantly longer. The model described in Appendix IV.A has taken three weeks to develop.

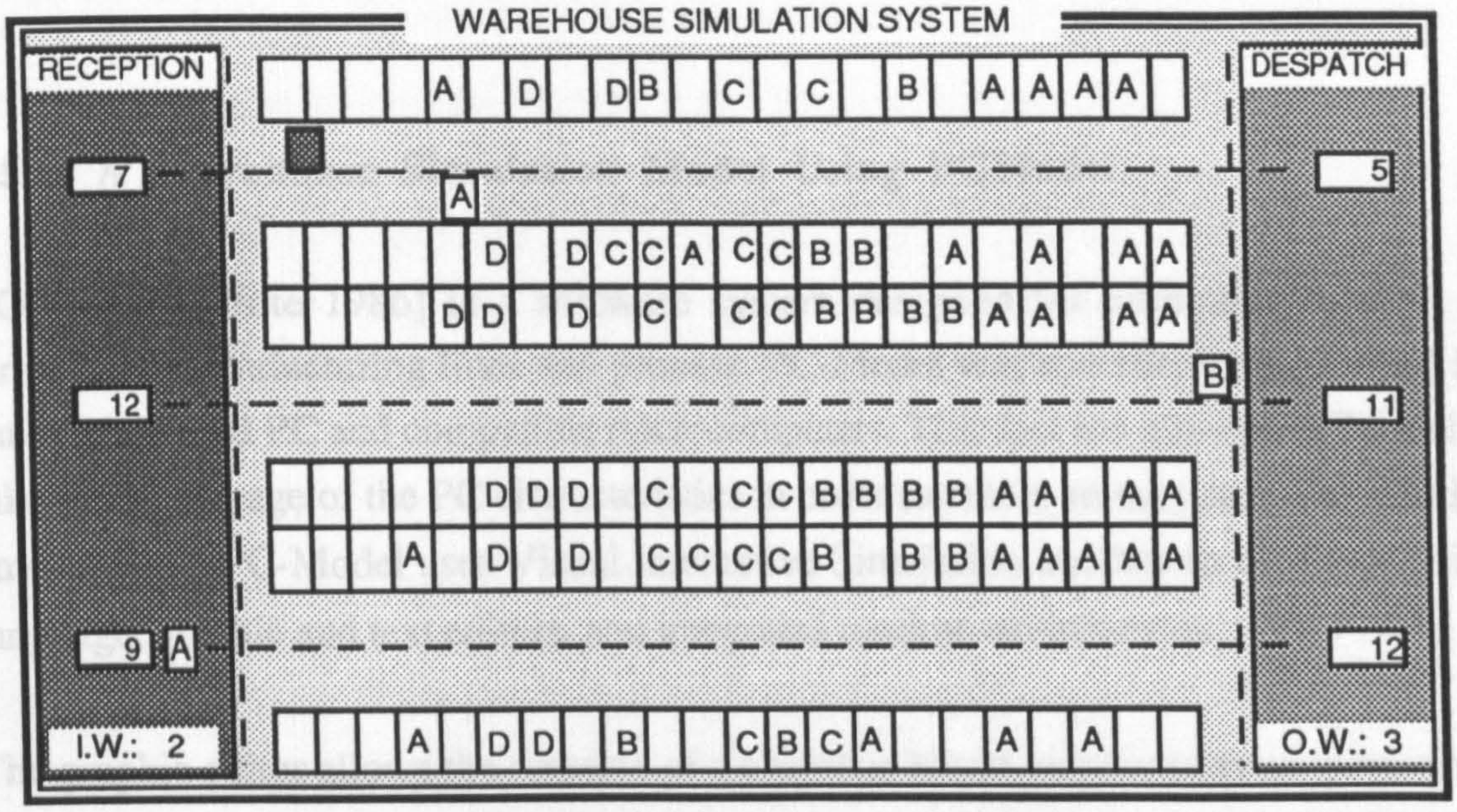


Figure 4.3 The Simulation Screen During the Running of PCModel Warehouse Model

PC-Model has been used to develop a warehouse simulation model. The system consists of a full pallet in/full pallet out one floor high warehouse. The schematic representation (overlay) of the warehouse is shown in figure 4.3.

The warehouse has three reception bays and three despatch bays. During the simulation the number of pallets waiting to be picked in the reception bays and the number of ordered pallets yet to be satisfied in the despatch bays are displayed near each bay (see figure 4.3). At the bottom of the reception and despatch areas the number of Inorders and Outorders (Trucks) waiting for a free bay are also displayed.

There are four different product groups named 'A', 'B', 'C' and 'D'. Each pallet is represented by the same letter of the corresponding product group. The product groups 'A', 'B', 'C' and 'D' represent 60%, 40%, 10% and 10% respectively of the total throughput volume. The warehouse has a capacity for 132 pallets. Each racking cell can store one pallet only. The pallets are put away in the free location nearest the despatch area and are picked following the rule FIFO (First In First Out).

There are two transporters, named PICK 1 and PICK 2, to unload pallets from the reception area to the storage area and two transporters, named REACH 1 and REACH 2, to load pallets from the storage area to the despatch area. The transporters are

necessary to build a detailed Visual Interactive Simulation (VIS) model for a complete warehouse system using a programming language and a simulation library.

4.5.1. A Warehouse Simulation Model Using PCModel

PC-Model [White 1986] is a software system designed for animating as well as simulating manufacturing lines and process. PC-Model was specifically developed for running on IBM PC and compatible microcomputers. This fact has allowed PCModel to take full advantage of the PC characteristics in order to create an easy to use simulation environment. PC-Model uses Visual Interactive Simulation and has its own modelling language, graphic and text editors, and integrated runtime environment.

The graphic editor allows the creation of a character based graphic screen representing the system to be simulated. The drawing is made in a column-row coordinate grid using the IBM-PC ASCII character set and a few graphic-primitive functions from PC-Model. The drawing, called overlay, is not limited to the physical screen 80 columns by 24 rows. A logical screen is provided with over 32000 column-row character cells. Alternatively the PC-Model/GAF (Graphic Animation Facility) version allows the use of high-resolution graphics generated with CADwrite or AutoCAD. The overlay acts as a background image over which the movement of characters or icons, called objects, can be made.

The model logic is defined by creating a text file with PC-Model language statements. This file can be edited within PCModel integrated environment with the text editor. The dynamics of the model are created by using the PC-Model language statements to create, move or delete objects. PC-Model handles automatically the movement of the objects and controls eventual congestion by not allowing more then one object to occupy an overlay location. The sequence of instructions describing the movement of an object and logical decision making are grouped in blocks known as routes.

PC-Model provides a series of interaction commands that can be executed during the running of the simulation. It is possible to pause, slow down or single-step the simulation run. The overlay image can be panned or if PC-Model/GAF version is used it can be zoomed. Model parameters and object attributes can be changed at any time. Statistics can be shown on the screen providing they have been defined in the language statements file.

must be made in the reverse order of the retailer's visit sequence to allow easy unloading.

The product cases throughput is very high. To cope with loading peak times and to prevent cases shortage EFACEC's engineers have designed a solution consisting of a set of cases buffering conveyors and depalletising tables (see fig. 4.4). The computer controls the opening of each buffering conveyor, according to the order's item list, allowing the cases to flow to the main conveyor and be loaded to the truck at a very high rate. When there is enough space in the buffering conveyors a pallet is automatically depalletised. Its cases are directed by a conveyor to the buffering conveyors that need to be replenished.

The depalletising tables are fed by a conveyor that accumulates pallets acting as a buffer. When a pallet is needed in this area the computer issues an order to the racking cranes that pick up the required pallet. The pallet is then taken to the depalletising area by the conveyor system. The depalletising tables and buffering conveyors are dedicated to certain products. The decision was taken according to the products throughput rate.

There is a total of thirty products. The demand is stable and seasonal effects are not significant. The Pareto analysis of the daily throughput is presented in figure 4.5. As can be observed, 20% of the products are responsible for 90% of the throughput.

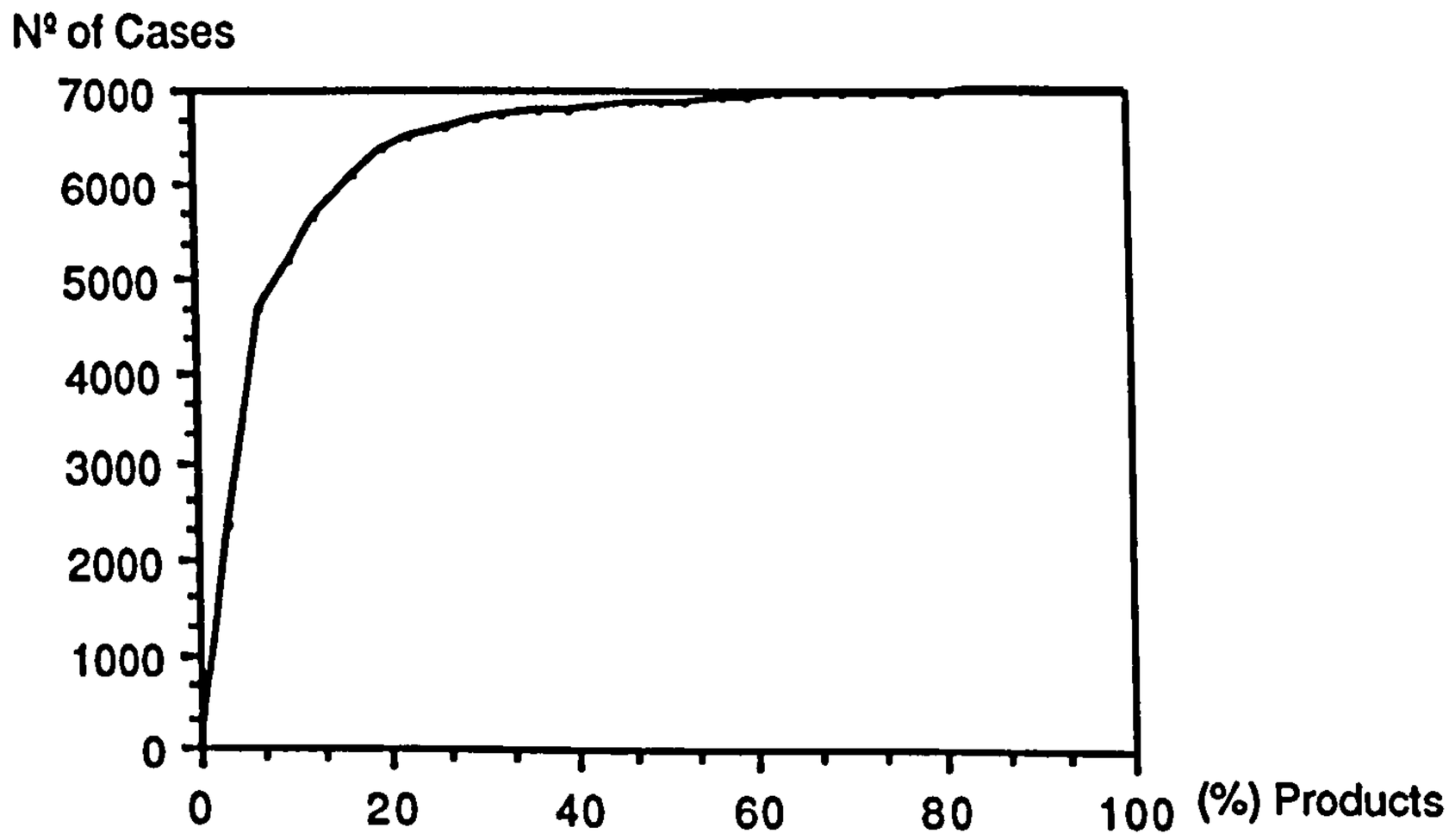


Figure 4.5 Pareto Analysis of the Daily Throughput

The products are grouped in five classes according to the daily throughput. Table 4.2 presents the class boundaries and the number of products in each class.

Class	Daily Throughput (cases)	Nº of products
1	> 1000	2
2	> 250	4
3	> 34	6
4	> 17	6
5	≤ 17	12

Table 4.2 The product classes in the EFACEC's warehouse

The twelve case buffering conveyors are dedicated to classes 1 and 2. Two case buffering conveyors and a depalletising table are dedicated to each product in class 1. One case buffering conveyor is dedicated to each product in class 2 and one depalletising table is dedicated to the whole class. Four conveyors and one depalletising table are left out for future expansion or to be used in the case of a break down.

The ten case buffering conveyors set is dedicated to classes 3, 4 and 5. One conveyor is dedicated to each product from class 3. The products from class 4 and 5 use two conveyors. Two conveyors are left out for future expansion or to be used in the case of a break down. The order's items for products in class 4 and 5 are prepared manually in advance and placed in the corresponding conveyor.

Products in classes 3, 4 and 5 are picked from the slow movers racking (see figure 4.4) and are handled manually due to its low rotation. The slow movers racking is replenished whenever a pallet is emptied. An order is issued to the racking cranes and the pallet is picked from the main racking and taken by the conveyor system to the slow movers zone. A transporter picks up the pallet at the conveyor end and puts it away in the racking.

The pallets are picked in the main racking using the FIFO (First In First Out) rule. This rule is used to prevent the products from being stored for long periods of time that could lead to deterioration. When the rail trolley has a pallet ready to be taken to the racking

the computer must decide to which aisle the pallet should go. The objective is to balance the number of product pallets in each aisle. The pallet is taken to the aisle which has the smallest number of pallets from that product. In each aisle the pallet is put away in the free cell that is nearest the aisle exit.

In the two EFACEC design options all the warehouse parameters are the same except the number of aisles and the racking dimensions. Table 4.3 shows the different parameters used in each option.

Option	Nº of aisles	Nº of pallets / level / aisle	Nº of levels	Nº of cells
1	6	156	5	4680
2	5	132	7	4620

Table 4.3 The number of aisles, levels and racking cells in the two design options

The inorders and outorders profile are well known by EFACEC's client. The products arrive to the warehouse in trucks coming from the company production plants. The outorders profile is defined based on the current distribution data.

4.5.2.2. Simulation Model

A Visual Interactive Simulation model was developed to evaluate EFACEC's design options. The model uses the SIMVIS libraries and utilities described in 5.3.1 and is developed in FORTRAN 77. It was first implemented in a Data General MV/8000 computer running AOS/VS operating system. Later on it was adapted to run on Digital VMS operating system for Digital VAX family of computers. In both versions a Tektronix 4109 graphics terminal is used to define and run the simulation model.

The development of the warehouse simulation model had the following objectives:

- to evaluate the warehouse overall performance;
- to verify whether the despatch bays and the cases buffering conveyor system were able to load the expected trucks in the required time and without delays;

- to establish the minimum number of racking cranes capable of doing the pallet storage and replenishment operations without bottle-necks;
- to convince the client of the quality of EFACEC's proposal;
- to be used as a competitive edge against competitors.

A scaled design, similar to the one from figure 4.4, is used to display the running of the simulation model. The pallets movement on the conveyors, the movement of the trolleys and racking cranes are realistic displayed. The individual pallets whilst being transported are represented by a scaled square which is draw using the product colour code. When the pallets are stored in the racking cell they are represented by a line segment with the same colour.

After the depalletisation (because of scale) it is not possible to do a detailed display of the cases flow. This flow is represented by a line which connects the table with the destination conveyor. The same technique is used to represent the case flow between the buffering conveyor and the truck being loaded. The colour and position of the line is updated every time a change occurs. An icon representation is used to show the trucks at the bays or waiting to get in.

The model structure is similar to the one described in 5.2.1. The configuration module allows, for the layout of figure 4.4, the definition of the coordinates, the number of aisles and the number of racking levels. In the IDUMP module the user is asked to define the start date and time and the duration of the simulation. Next it generates the internal simulation model structure.

The simulation module allows the running of any previously defined configuration. Figure 4.6 shows a picture of the display during the running of the simulation model. The running of the simulation can be interrupted at any time and the user can execute several SIMVIS commands called system interactions (see 5.2.1). The user can also execute any of the following commands developed specifically for this model and called user interactions:

INFSTO - transporters histogram statistics;
ESTATI - current warehouse state data;
INFCHA - trolleys histogram statistics;
INFPRO - histogram of the nº of pallets for each product / all products;
STOCK - change the transporter's control logic.

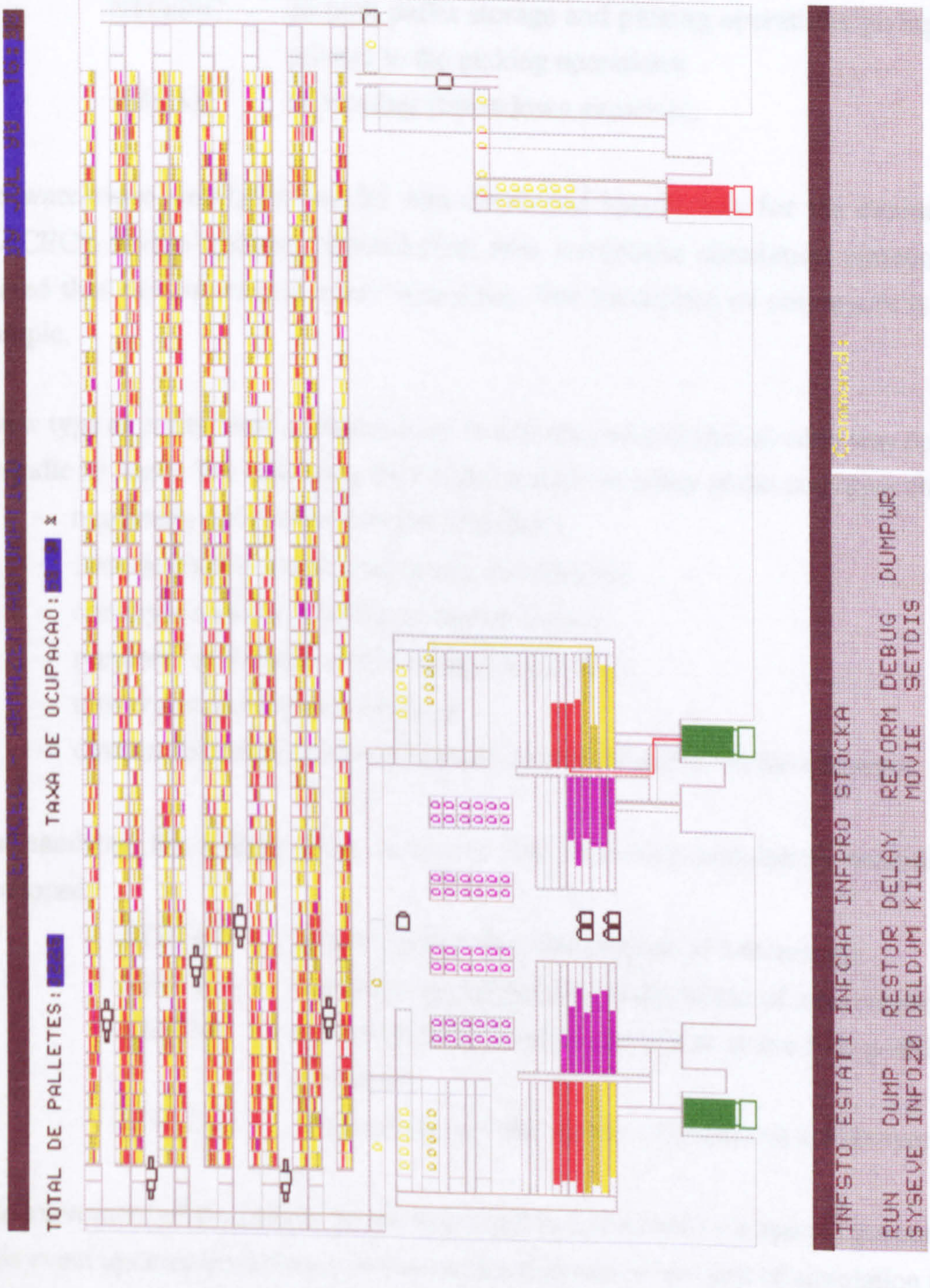


Figure 4.6 Screen display during the running of the "EFACEC" model

The STOCK interaction allows the user to change at any time the transporter's control logic. Any of the following rules can be defined for each one of the racking cranes:

STORE - do only pallet storage operations;
PICK - do only pallet picking operations;
STO/PIC - do both pallet storage and picking operations giving higher priority to the picking operations;
BREAK - do nothing (breakdown situation).

The warehouse simulation model was developed specifically for the evaluation of EFACEC's design options. Nevertheless new warehouse simulation elements were created that can be used in other situations. The modelling of conveyors is a good example.

A new type of entity class named conveyor was created and special code was developed to handle its logic. The following data is defined for an entity of the conveyor class:

- maximum and current number of pallets;
- conveyor's start and end physical coordinates;
- conveyor's speed, width and display colour;
- pointers for the pallet's source and destination;
- movement and breakdown flags;
- circular list and the required data to access the pallets on the conveyor.

For handling the pallets on a conveyor the following simulation modules were developed:

ADITAP - adds a pallet to the start position of a conveyor;
RETTAP - removes a pallet from the end position of a conveyor;
LEPTAP - retrieves the pointer to the pallet at the Nth position on a conveyor;
LNPTAP - retrieves the current number of pallets on a conveyor.

The movement of the pallets on the conveyor is controlled by a special purpose event. This event updates the pallets position on the conveyor every unit of simulation time and schedules itself while necessary. It also schedules a user defined event when a pallet reaches the end of the conveyor. Two different types of conveyors are included. One

stops the movement when a pallet reaches the end and it is blocked. The other moves the pallets and accumulates them at the end of the conveyor.

4.5.2.3. Conclusions

The objectives described in 4.5.2.2. for the development of the EFACEC's simulation model were achieved. The running of the simulation model has given a thorough understanding of the warehouse design options.

A major objective was to decide between design options 1 and 2 (see table 4.3). As the crane cost is high the option 2 was preferable. The question was to know if five cranes were enough to do the job. The simulation study of a typical 8 hours shift showed that five cranes were capable of doing the pallet storage and replenishment operations without delays. Figure 4.7 presents the cranes statistics for a 4, 5 and 6 aisles options in a 8 hours shift.

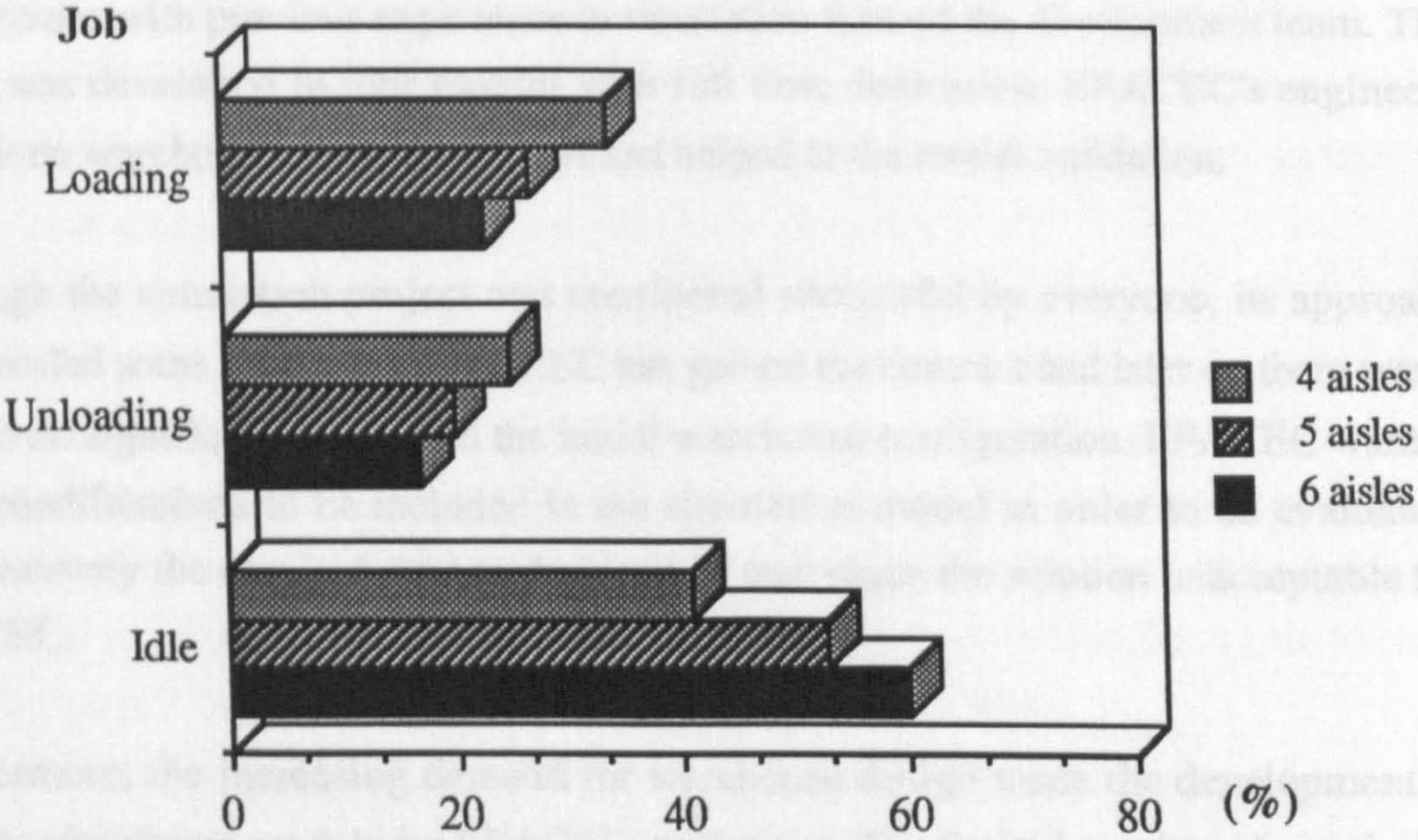


Figure 4.7 Cranes operational statistics (% of simulation time)

As it is necessary to allow for a crane to be stopped for maintenance without affecting the pallet replenishment operations the four crane option was also analysed. The

simulation results showed that five cranes is sufficient for the replenishment operations with allowances for maintenance and future increase in the demand.

The simulation study has also demonstrated that the despatch bays and the case buffering conveyor system are capable of loading the expected trucks in the required time and without delays. A truck waiting to be loaded queue was never formed.

The depalletising tables and the pallets replenishment system have proved to be effective. Under normal working conditions the trucks loading cases flow was never interrupted by cases shortage. The simulation analysis of the warehouse overall performance was fully satisfactory.

The model was also used to explain the solution to EFACEC's client and to convince him of the merit of the project. The scaled animation and the quality of the graphics display have caused a great impact in the client. The fact that EFACEC was the only company to use simulation to evaluate the project was used as a competitive edge against competitors. The simulation project was fully successful.

Two persons with previous experience in simulation formed the development team. The model was developed in four months with full time dedication. EFACEC's engineers have given warehouse technical support and helped in the model validation.

Although the simulation project was considered successful by everyone, its approach has revealed some limitations. EFACEC has gained the contract and later on there was a need to do significant changes in the initial warehouse configuration. EFACEC wanted these modifications to be included in the simulation model in order to be evaluated. Unfortunately the required time to do it and its cost made the solution unacceptable for EFACEC.

Furthermore, the increasing demand for warehouse design made the development of specific simulation models for EFACEC inadequate. The limited number of simulation experts and the required development time made this approach impractical. EFACEC knew that simulation was crucial in warehouse design but it has no internal expertise and people to do it. So other alternatives had to be looked for.

4.6. Conclusions

4.6.1. Current Alternatives for Warehouse Simulation

When a warehouse simulation model has to be developed two options can be made: use a consultant or simulation expert; or develop the model within the warehouse management or design team.

The use of a consultant or simulation expert can speed up the process of obtaining results and gives professional expertise. However, this option is usually more expensive and most of the time does not have the flexibility required. Also, communication problems can arise between the simulation expert and the warehouse team. The simulation expert is not aware of warehouse problems and the warehouse team have difficulties in understanding some technical aspects of the simulation. These difficulties are a major drawback and can compromise the success of simulation projects.

Furthermore, with the growing need of simulation models in warehousing it is not possible to find always a simulation expert available to do the job. The development of the simulation model within the warehouse team appears so to be the most desirable solution. The question is: how appropriate current simulation systems are to be used by people from the warehouse business to develop warehouse models?

In a warehouse management or design team it is not very common to find someone with simulation modelling expertise or programming experience. So any simulation system that requires a reasonable knowledge of simulation concepts or requires programming skills for developing the model will not be realistic to use. At least for acceptable development times. Even a simple warehouse model developed using a user friendly package such as PCmodel (see 4.5.1) can become quite complex.

For a simulation system to be effectively used by warehouse designers it must have a user friendly interface, a warehousing specific terminology and it must be easy to use and enable the development of simulation models in short periods of time. From all the different simulation approaches described, the most adequate to provide these characteristics is the Data-driven Generic Model approach. These aspects were already recognised in the manufacturing systems area. In the last decade several simulation systems using this approach were developed for the manufacturing systems area.

Manufacturing and warehousing systems have some similarities. The existing manufacturing simulation systems can sometimes be used to build warehouse models. However common elements such as transporters or storage units have different contributions for the global efficiency of manufacturing and warehousing systems. Movement is the prime activity in a warehouse; machining processing is the prime activity in a production unit.

The manufacturing simulation systems can be used by warehouse management or design teams for a first approach in the evaluation of a warehouse system or to model specific areas within a warehouse. However, they are not capable of handling all the complexity of a warehouse system and cannot represent with an acceptable level of accuracy the relationship between all the warehouse elements.

The current simulation systems that could be used by warehouse business people are not appropriate to simulate warehouses. To model the complex logic of a warehouse system a programming or simulation language has to be used. For that, expertise in simulation and programming are required. This kind of expertise is not common to find within warehouse business people and the increasing need for warehouse simulation makes each time more difficult to find external consultants to do the job. In either case the development of dedicated warehouse simulation models is very time consuming and does not have the flexibility required (see 4.5.2.).

The Data-driven Generic Models approach is the right one to allow warehouse designers to build simulation models. However current simulation systems using this approach are not appropriate for warehouse simulation. Warehouse systems have special characteristics that require a new way of defining the simulation model and new modelling elements to represent its complex logic.

4.6.2. Design Brief

As stated above manufacturing systems share some common elements with warehousing systems (e.g. transporters, storage units, etc). However these elements influence the manufacturing and warehousing system's behaviour in different ways.

In manufacturing systems machine working is the centre of activity and transport and storage are auxiliary activities. In manufacturing systems job processing times and

CHAPTER 5

AWARD AND THE EVOLUTION OF A DATA-DRIVEN GENERIC MODEL WITH CAD INTERFACE

scheduling rules must be modelled accurately while travel times can be taken approximately, storage access logic simplified and physical locations ignored.

In warehousing systems movement is the centre of activity. The system performance depends mainly on the efficiency of the storage and picking operations which in turn depends on the speed in which pallets are moved, picked and put away. As movement is the prime activity in a warehouse: storage physical locations, travel paths, transporter's speed must be rigorously modelled. This explains why current manufacturing simulation systems are not appropriate for warehouse simulation.

Physical locations and travel distances are crucial in warehouse modelling. The warehouse layout is a key factor in the success of the system. When developing warehouse models the spatial factor is essential for a good representation of the real system.

A warehousing simulation system using the Data-driven Generic Model approach should then have the following characteristics:

- a user friendly interface using warehouse terminology;
- an effective way of defining the physical layout in order to enable the model of movement with the required level of accuracy;
- simulation elements which allow a correct modelling of warehouse logic.

The next chapter describes the evolution of AWARD (Advanced WARehouse Design) a software package to support warehouse design and management. AWARD development has been user oriented and incorporates the experience and background discussed above.

PAGE
MISSING IN
ORIGINAL

5. AWARD and the Evolution of a Data-Driven Generic Model with CAD Interface

5.1. Introduction

The Management Group (GEIN) from the Mechanical Engineering Department of Oporto University - Portugal had been doing since 1982 significant work in Visual Interactive Simulation (VIS). This work started with [Moreira da Silva 1982] following his experience in VIS at Warwick University. In the first stage a set of basic software tools necessary for the implementation of VIS models were developed. The decision to develop a simulation system rather then use commercial software packages has resulted from the fact that hardware availability, efficiency and flexibility were key factors at that time. The system developed SIMVIS [Bastos & Moreira da Silva 1985] is an event-based visual interactive general purpose system and was first written in FORTRAN 77 in a MV/8000 Data General Computer for a semi-graphic ISC intecolor 4300 terminal. SIMVIS had a main simulation subroutine library and some software development aiding tools like an I/O subroutine library [Brito & Bastos 1985] and an interactive data-entry screen generator (MSC) [Brito 1985].

Although SIMVIS was used to build simulation models its main objective was the development of what can be called [Pidd 1988] Data-Driven Generic Models (see 1.2.). The SIMVIS author's strategy was to conceive a system with the basic tools necessary to support the development of model generators for specific domains. That strategy was justified because of the absence of packages with the same purposes and also, because of the model generators benefits for the end users. With the Data-Driven Generic Models the user can build models with a friendly interface using a dialogue that he can understand and without a great need of simulation expertise.

Several Data-Driven Generic Models for different areas were developed at GEIN using SIMVIS. Some examples of those are:

JOBSHOP - to build simulation models for the analysis of job-shop planning problems [Bastos 1985];

VISUALPLAN - to build simulation models for the design, planning and control of flexible manufacturing systems FMS [Moreira da Silva & Bastos 1984]; a project developed for the

company Asea-Brown Boveri (ABB), Baden, Switzerland;

SIMAHID - to build simulation models for evaluating and testing control rules and personal training in the operation of sets of interdependent hydroelectric power dams [Guimarães et al. 1985]; a project developed for the company Electricity of Portugal (EDP).

The Distribution Studies Unit (DSU) - Cranfield has been involved, since 1980, with teaching and consultancy in the area of warehouse design. From its experience in simulation studies of warehouse systems the DSU knew the importance of this technique, especially for the analysis of complex and large systems. However, the DSU was also aware that existing simulation software was difficult and time-consuming to use. Previous contacts between GEIN - University of Oporto and DSU - Cranfield identified, in early 1986, the need for a software package which could help warehouse designers and managers to design, test and evaluate different alternatives. The package was named **AWARD** Advanced **WAR**ehouse **D**esign and a first prototype was developed during 1986.

The need for technical and financial support to develop AWARD led to contacts with companies from the materials handling and computer industries. In January 1987 a first meeting [DSU et al. 1987] was held at the DSU - Cranfield where the AWARD concept was discussed and the use of the prototype demonstrated all its potential. After a second meeting, held in February, an AWARD consortium was formed by the following members:

- INEGI, University of Oporto;
- DSU, Cranfield Institute of Technology;
- Coopers and Lybrand Associates;
- National Carriers Contract Services Limited;
- EFACEC S.A.R.L., Portugal;

- Digital Equipment Corporation;
- Dexion Limited;
- Synergy Distribution Systems Limited;
- John Lewis Partnership.

The creation of the AWARD consortium had gathered the necessary conditions in both technical input and financial support to do the research work and development of the AWARD package.

5.2. Software Design Methodology

5.2.1. Previous Work Concepts and Structure

Since the beginning, SIMVIS development was oriented to support visual interactive simulation systems that could be easily understood by end users. This concept is very important because normally, the people who best know the system to be simulated (usually the decision makers) are the ones that can give major contributions to the development of the model and to the analysis of the results. Quite often those people have difficulties in understanding the technical aspects of the simulation process and can become sceptical about it; this can compromise the success of the project. To overcome that, the fundamental principles behind the creation of Data-Driven Generic Models using SIMVIS are:

- the use of a front end, with a friendly interface, within which the user can define the problem in his own technical language without a significant simulation knowledge;
- the automatic generation of the model, based on the user input data, so that complex simulation details can be hidden from the user;
- the extensive use of colours, graphics and animation so that visual aspects can help the user to validate the model and see why problems arise during its running; and interpreting the simulation results;

- the use of interactive facilities to quickly change model parameters, obtain further information about the state of the model and to display statistical reports.

All the Data-Driven Generic Models developed with SIMVIS share a common software structure (see figure 5.1). That structure is formed by the following modules:

CONFIG - the configuration module, where the user defines the physical characteristics, the logical relationships and the initial conditions of the model;

IDUMP - the initial dump module, where the internal simulation structure is created from the data defined in the previous module;

MODEL - the simulation module, where the different models can be executed.

. Configuration module

The definition of a particular system to be simulated is made using a data-entry menu-driven program called CONFIG. A friendly user interface is used so that no special computer skills are needed. The data-entry is organized in a series of screens of related information. In each screen the user can move forwards and backwards from one field to another and modify the field contents as he wishes. A field validation is made in the aspects of: the field type (e.g. integer, real or character); the field limits (e.g. a bounded numerical value); and field specific words (e.g. "Yes" or "No"). The user can go back to the previous screen or to any other screen through the use of menus so that he can input or change data at any time and in any order he likes.

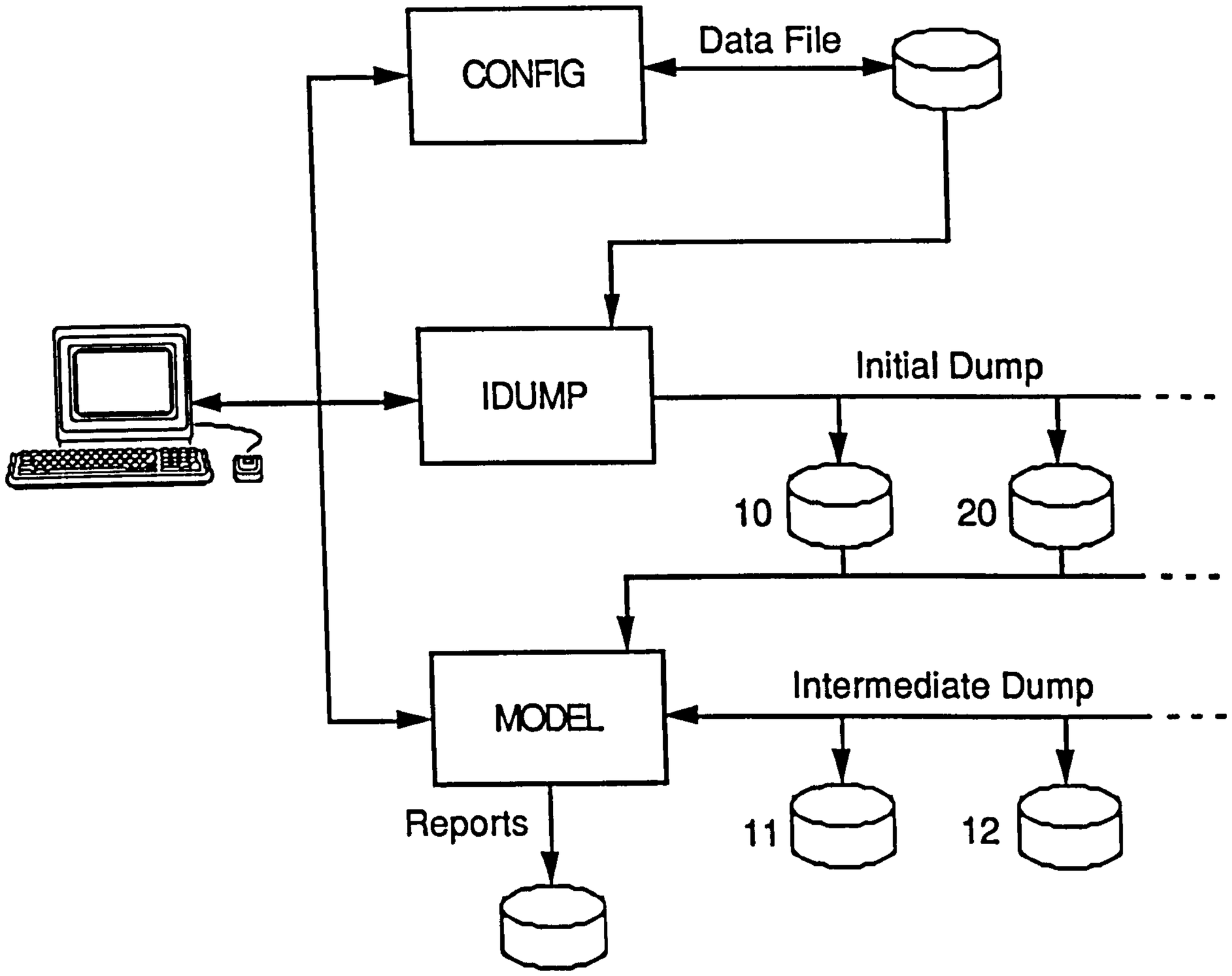


Figure 5.1 Structure of a SIMVIS Data-Driven Generic Model

To each problem is associated a data file, named by the user, that can be edited to change system parameters or it can be used as a starting point to define similar problems. There is, usually in the end of each session, an overall data check to ensure that a coherent set of data was defined and to give some guarantees that will be possible to create the corresponding simulation model. The terminology used in the data-entry screens is the same as that used in the area for which the generic model is developed. So any person, within that area, should have no difficulties in understanding the type of information required.

. Initial dump module

The data defined at the configuration stage is used by the program IDUMP to build the internal simulation structure. This program normally runs on the background without the user intervention. The simulation elements for each configuration are created using

the SIMVIS simulation library subroutines. The most important element types available in SIMVIS [Bastos & Moreira da Silva 1985] are the following:

Entity - system element which can be individually identified and processed; each entity has, usually, a set of attributes which contain numeric or alphanumeric information;

Group - set of entities (e.g. class of entities with the same set of attributes and a common name);

Queue - physical or logical place where entities are stored;

Logical display - system element that contains the relevant display information about the state of the model; it works as a transparency that can be set ON or OFF, allowing a modular representation of screen information;

Histogram - system element where statistics collected during the execution of the model are stored and maintained.

Program IDUMP creates all the simulation elements necessary for each particular model and also makes the set up of the initial conditions. The main tasks handled by IDUMP are the creation of the entities, groups, queues, logical displays and histograms, the definition of the relevant attributes of all simulation elements, the placement of the entities and group of entities in the right queues and the scheduling of the initial events. All this is made using the subroutines from SIMVIS library, for a complete description see [Bastos & Moreira da Silva 1985].

At this stage some error checking is performed and if something wrong is found, then an adequate message is given to the user. Normally that is due to inconsistent data. The habitual procedure will be to return to the configuration module, make the required changes, and then come back to IDUMP. If no errors are found then a copy of the model structure in memory is dumped to a disk file. Several initial dump files can be created for the same problem and to each of them is associated a different number (see fig. 5.1). The user can select what initial dump file will be used to start running the model.

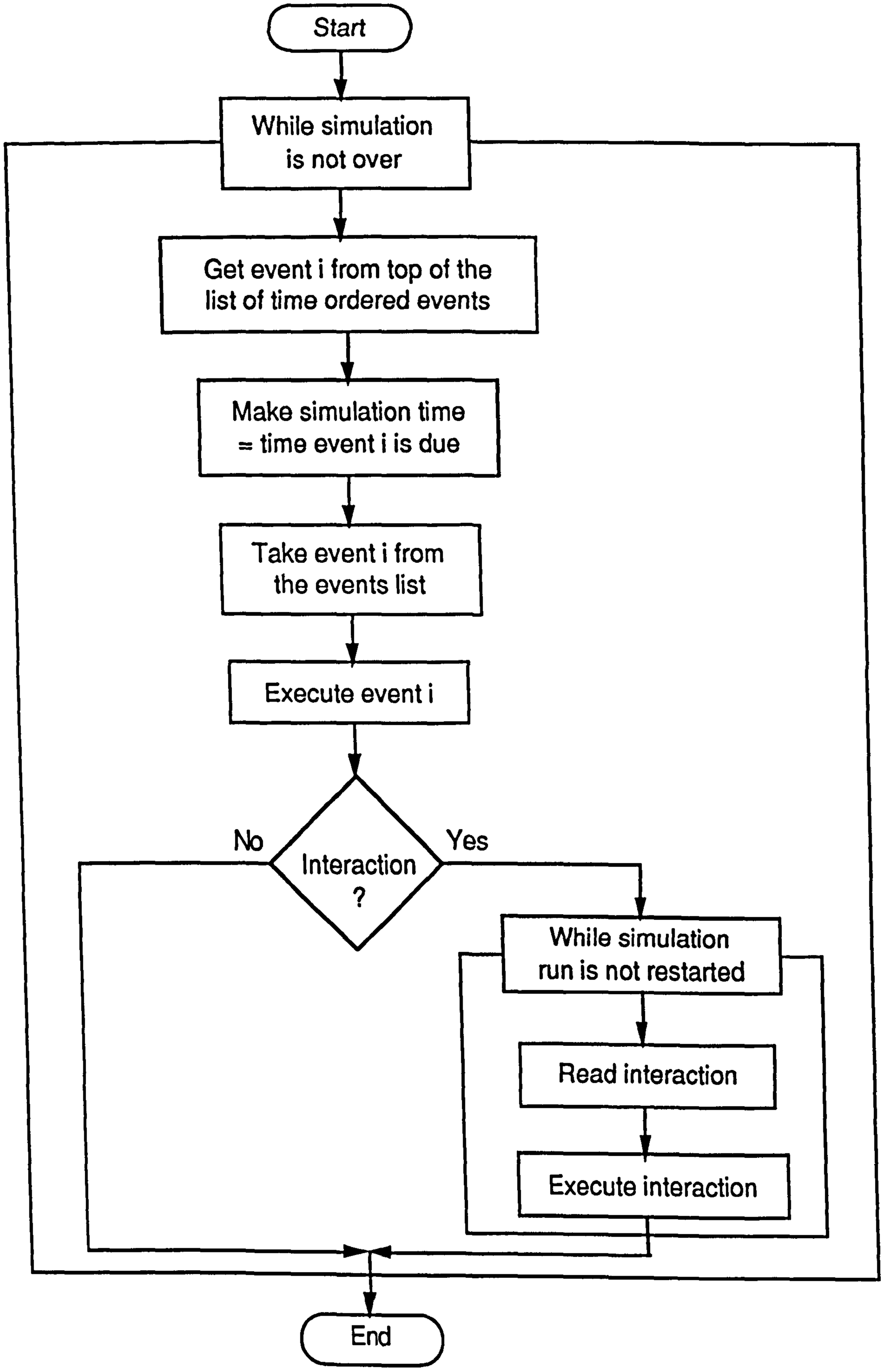


Figure 5.2 Logic diagram for the SIMVIS executive

. Simulation module

The program MODEL follows the general structure for discrete modelling described by Fishman [Fishman 1973] (see 2.2.1.). The executive is implemented using the next event technique (see 2.1.) and the event approach modelling (see 2.2.2.). As SIMVIS is an interactive simulation package, special logic was included to handle user interrupts. However the simulation run can only be interrupted between the occurrence of events. After the simulation has been interrupted the user has access to a set of available commands called interactions.

The user can execute any number of interactions and then restart the simulation run (see fig. 5.2). The generic system for each application area is modelled through a set of events. The way these events are developed is very important because it will establish the flexibility of the generic model. For a particular problem and for a set of initial conditions the system behaviour is reproduced by the execution of a sequence of events controlled by the executive (see fig. 5.2).

The simulation run can start from one of the initial dumps created by IDUMP or from an intermediate dump created previously during model execution (see fig. 5.1). An icon image of the real system is continuously updated on the screen while the model is running. That represents a great help in validating the model and also in understanding why problems appear. The user at any time can interrupt the model and execute one of the available interactions. These interactions are divided in two groups: the system interactions; and the generic model interactions. The system interactions perform standard functions which are common to the various generic models. The generic model interactions are specific to each application area and usually have a special significance within that area. Some of the major system interactions are the following [Bastos & Moreira da Silva 1985]:

RUN - restart the simulation run;

KILL - stops the model run and end the simulation session;

DELAY - delays the model execution for a user defined value;

DUMP - copy the internal memory simulation structure to a disk file called intermediate dump file (see fig. 5.1);

RESTOR - restore the internal memory simulation structure from a previous saved intermediate dump file (see fig. 5.1).

The DUMP command is very useful because it permits the saving of a particular model situation to a disk file and later, using the command RESTOR, the restarting of the simulation from that point.

5.2.2. AWARD Concepts and Structure

The developments in graphics terminals and the decrease in their cost, has made possible many new applications. Visual interactive simulation is an area where the use of graphics has obvious benefits. The use of colours and higher resolution graphics, in displaying the image of the model running, can help the user to distinguish the different types of information and to have a better understanding of what is happening. Bearing this in mind the SIMVIS package was adapted to take advantage of the graphics facilities from Tektronix 4109/4107 terminals. A new graphics library was developed [Brito 1984] and changes were made to the existing ones in order to incorporate the new terminals.

The first generic model using the Tektronix terminal was SIMAHID [Guimarães et al. 1985]. Graphics were used at the configuration stage to show the hydroelectric power dams network so that the user could easily check the input data. During the model execution icons were used in a pictorial representation of the dams. In a SIMAHID - ZOOM interaction it was possible to observe the functioning of a particular dam. At this level of detail it was possible with special conceived dynamic XY graphics to follow the behaviour of some of the most important state variables. This was essential for the analyst to be able to control the dams during floods.

In early 1986, when discussing the first ideas of a data-driven generic model for warehouse design, some important aspects were raised. One of them was the necessity of an accurate representation of the materials handling equipment. This was mentioned not only to the travel times but also to the traffic congestion problems. Another important aspect was the layout. How to define the warehouse layout in order to be rigourous in representing the racking location, product zoning, transporters paths, reception and despatch areas etc.. Furthermore, how to do all this and keep it simple

and easy to use in order that it could be an aid to warehouse designers and managers without computing or simulation experience.

The development of a prototype was the first step towards a possible solution to some of the mentioned problems. Following the work done it was decided to develop a data-driven generic model for warehouse simulation. The only way that seemed reasonable enough to cope with the layout detailed definition was the use of a warehouse scale drawing. This would not only provide the required detail but as well it would supply a way to define the transporter paths. During 1986 a prototype was developed with the same structure as the SIMVIS data-driven generic models (see fig. 5.1).

In the prototype configuration module the warehouse data was entered using the same method as before. A sequence of data-entry screens guided the user in the input of the necessary information to define the warehouse system. The new concept introduced here was the drawing of the scaled warehouse plant in parallel to the data input. This was very useful because the user could do an immediate visual validation of the data entered. Although this new approach was well received by the users it presented some difficulties in what concerned the time consumed in getting all the numerical information needed. Furthermore, if the wrong figures were introduced in one data screen and the user went to the next data screen there was no facility to go back and correct the error. When this happened the only way to correct the error was to restart the configuration. Figure 5.3 gives a brief description of the type of data needed and shows the kind of graphics displayed in the background during the configuration stage.

The internal simulation model structure was, as in the other SIMVIS generic models, created by the IDUMP program (see fig. 5.1) and without user intervention. In the simulation module the warehouse scaled design was used to display the model running. At this stage it was possible to follow the movement of the transporters along the warehouse. Their main tasks were: unload the trucks at the reception, and take the pallets to the racking; and load the pallets from the racking and take them to the trucks waiting at the despatch area. Their movement was made along the right side of the aisle and the shortest path was chosen between the initial and final points. Transporter

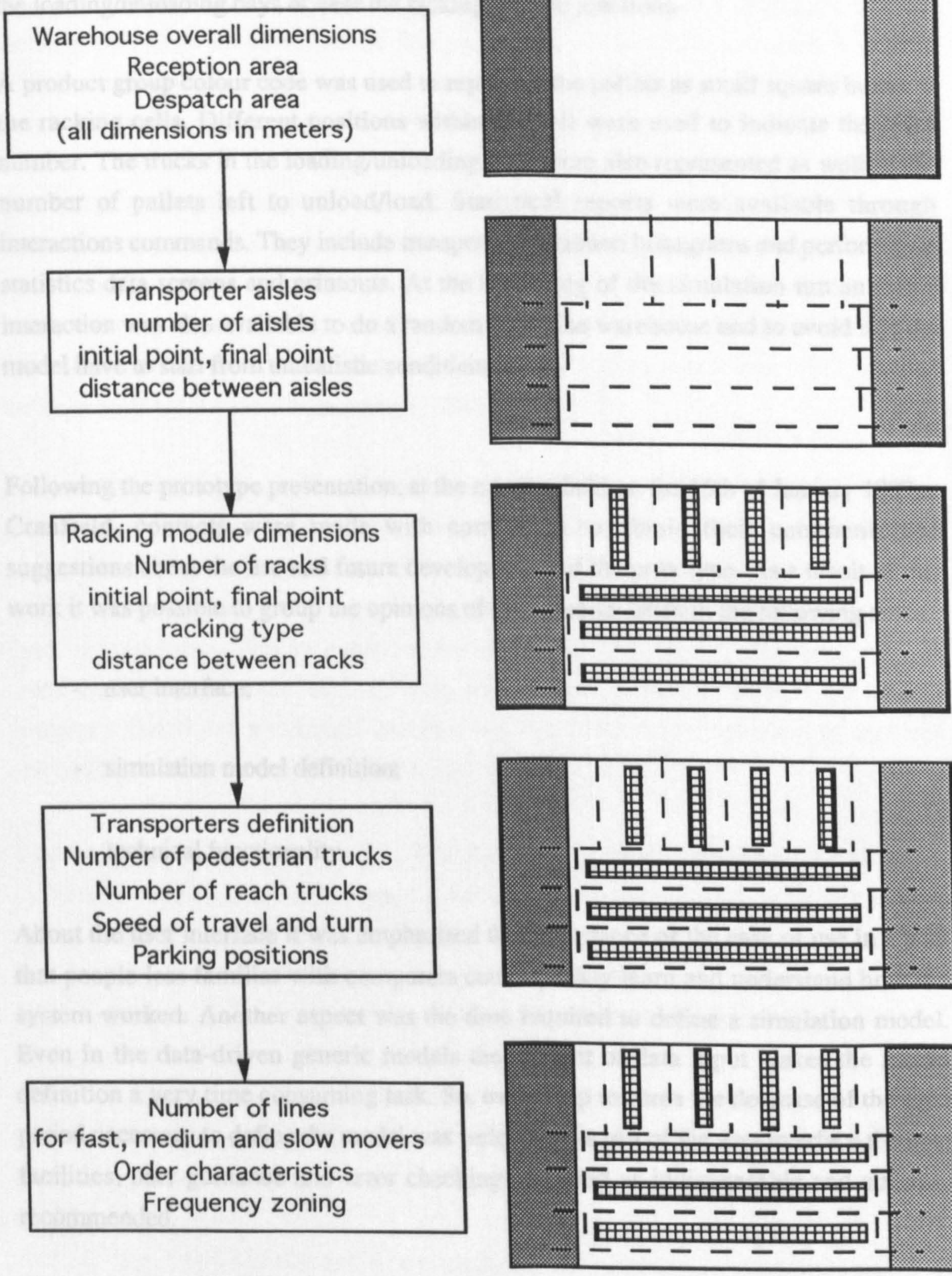


Figure 5.3 Sequence of input and graphics background in AWARD prototype

congestion was realistically modelled and so the user could see queues being formed in the loading/unloading bays or near the racking or aisle junctions.

A product group colour code was used to represent the pallets as small square boxes in the racking cells. Different positions within the cell were used to indicate the level number. The trucks in the loading/unloading bays were also represented as well as the number of pallets left to unload/load. Statistical reports were available through interactions commands. They include transporter utilization histograms and performance statistics data screens and printouts. At the beginning of the simulation run an initial interaction was also available to do a random fill of the warehouse and to avoid that the model have to start from unrealistic conditions.

Following the prototype presentation, at the meeting held on the 15th of January 1987 at Cranfield, contacts were made with companies to obtain their comments and suggestions about the use and future developments of the prototype. As a result of this work it was possible to group the opinions of the potential users in the following areas:

- user interface;
- simulation model definition;
- technical functionality.

About the user interface it was emphasised the importance of the ease of use in a way that people less familiar with computers could quickly learn and understand how the system worked. Another aspect was the time required to define a simulation model. Even in the data-driven generic models the amount of data input makes the model definition a very time consuming task. So, every step towards the decrease of the time period necessary to define the model was welcome. As part of the user interface the help facilities, user guidance and error checking appeared as indispensable and strongly recommended.

The technical details of the model definition should be, as much as possible, hidden from the user. The intention here was to be able to define the simulation model using a terminology known to the user and through a series of logical steps in a way that the user should feel comfortable. The error checking assumed an important role during the

model definition because it could save a lot of time, in a later stage, when it comes to debug the model. Another important issue was the overall efficiency. Although with the increase in computer performance this seemed to be less of a problem it is still fundamental when it comes to visual and interactive (a few seconds could seen an eternity when you wait for the computer to reply).

The technical functionality of the user opinions differed depending on the type of business their companies were involved in. This led to a very large number of different options. Some of them could be treated in a similar way by the simulation and others were too specific to be considered. Even so, it was necessary to define what was the minimum functionality to be included in a first stage and try to prioritise what was left out in order that this could be handled in later stages.

After making the analysis of the information available, the objective was to find the best way to solve the problem and at the same time meet the user expectations. The existence of the prototype proved to be very useful because it had worked as a starting point for the future developments. The use, in the prototype, of a scaled graphic drawing during the configuration, although a good concept, still presented some drawbacks. These were, mainly, due to the way the data was defined. Looking to the type of data necessary to define a warehouse system it was found that most of it could be captured graphically.

This had great advantages because it would make it much easier and speed up the warehouse configuration task. For implementing this concept the natural choice was the use of CAD (Computer Aided Design) in one of the following two ways: choose a commercially available CAD package and build an interface with the simulation; or use CAD techniques to develop a dedicated system for warehouse design.

The more straight forward solution and the one that would have involved less work was the use of a CAD package. But even if this option appeared attractive and had some advantages it was impractical mainly due to the following reasons:

- the cost; usually a CAD package is very expensive and it wouldn't be economically viable to buy it just to be use as a data entry tool for a warehouse simulation package;

- the level of training required; because of the great flexibility and power resulting in being general-purpose, the use of a CAD package is quite complex and normally needs long learning periods;

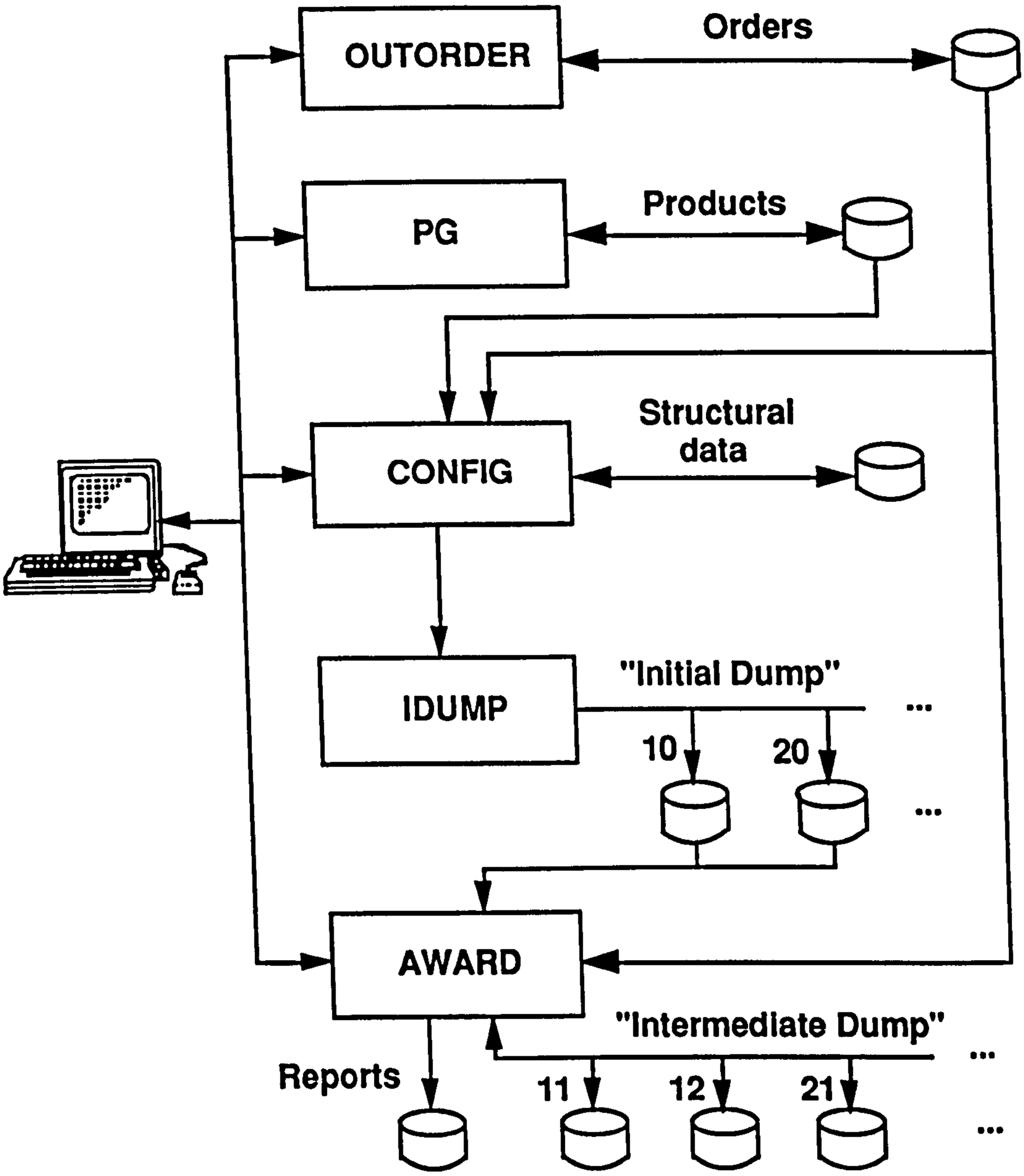


Figure 5.4 AWARD software structure

Hence, it was decided to substitute the screen oriented data-entry module of the prototype by a dedicated CAD based module where most of the warehouse system information could be defined graphically. This was the major new concept introduced to solve problems raised by the users concerning difficulties in the use of the prototype. Nevertheless to increase the productivity during the configuration stage others aspects

were covered as: a new menu system; flexible editing facilities; internal calculations and assumptions in order to define data automatically; improved validation, error checking and error messages.

Figure 5.4 shows the logic diagram for the AWARD software structure. This structure is similar to the one used in other SIMVIS data-driven generic models (see figure 5.1). The simulation model definition is made by IDUMP (see figure 5.4) which the user can execute from a menu option in the configuration module CONFIG. This option can only be selected after all the warehouse system data is introduced. This data included not only the information directly related to the warehouse itself but also to the products and orders. The data files for products and orders can be created, or edited, with a normal editor or using the interactive modules OUTORDER and PG (see figure 5.4).

With the approach described the user is completely unaware of the details related to the building of the simulation model. Furthermore the problem is specified using a friendly interface (CONFIG) with a terminology that is familiar to him. During the model definition IDUMP does a consistency data check to prevent as many errors as possible occuring later, when the simulation is running. Another important issue behind the conception of AWARD was the concern about the efficiency of the algorithms used. Being an interactive and graphical system the performance was essential for the success of it. The technical options included in the first version of the system were considered by all as the minimum to be realistic. The way the system was to be developed was such that new options could be added without major modifications.

5.2.3. New Concepts in AWARD

AWARD structure although similar to other SIMVIS data-driven generic models introduces new concepts in the way the simulation model is defined. In the data-driven generic models developed with SIMVIS the model was defined by entering data in a sequence of screens. In AWARD, Computer Aided Design (CAD) is used to define most of the model data graphically. This enables the capture of the warehouse data with the required detail (scaled design) to define the simulation model. Most of the warehouse objects (e.g. racking, transporters, aisles, etc.) are related directly with simulation elements.

5.3. AWARD System Description

5.3.1. SIMVIS Libraries and Utilities

5.3.1.1. Overall Organization

The SIMVIS libraries are organized in a way that they can be easily ported to another operating systems or adapted to work with different terminals. This is achieved by building separate libraries for the low level I/O and graphics subroutines, that are device and/or operating system dependent, and for the high level I/O and simulation subroutines, that are device and operating system independent. Figure 5.5 shows the overall organization of the SIMVIS libraries. If SIMVIS needs to be transferred to another operating system but the same graphics terminal is going to be used, only the low level I/O library STDIO needs to be changed.

Also, only the graphics library TEK needs to be changed if a different graphic terminal is going to be used in the same operating system. Both libraries need to be modified if simultaneously there is a change in the operating system and in the graphics device. SIMVIS is formed by the following FORTRAN 77 subroutine libraries:

. **STDIO** - [Brito & Bastos 1985] low level I/O subroutines to handle functions like: cursor control; binary read/write character by character; define character attributes (colours, size, underline, blink ...); clear all/part of the screen;

. **TEK** - [Brito 1984] graphic subroutines for interfacing with Tektronix 4107/4109 compatible terminals;

. **INSTA** - [Brito & Bastos 1985] screen management and high level I/O subroutines to handle functions like: read/write real, integer and character values with validation and screen attributes; input/output from/to screens created with the interactive data-entry screen generator MSC [Brito 1985] package;

. **SIMVIS** - [Bastos & Moreira da Silva 1985] simulation subroutines to support the development of visual interactive simulation models.

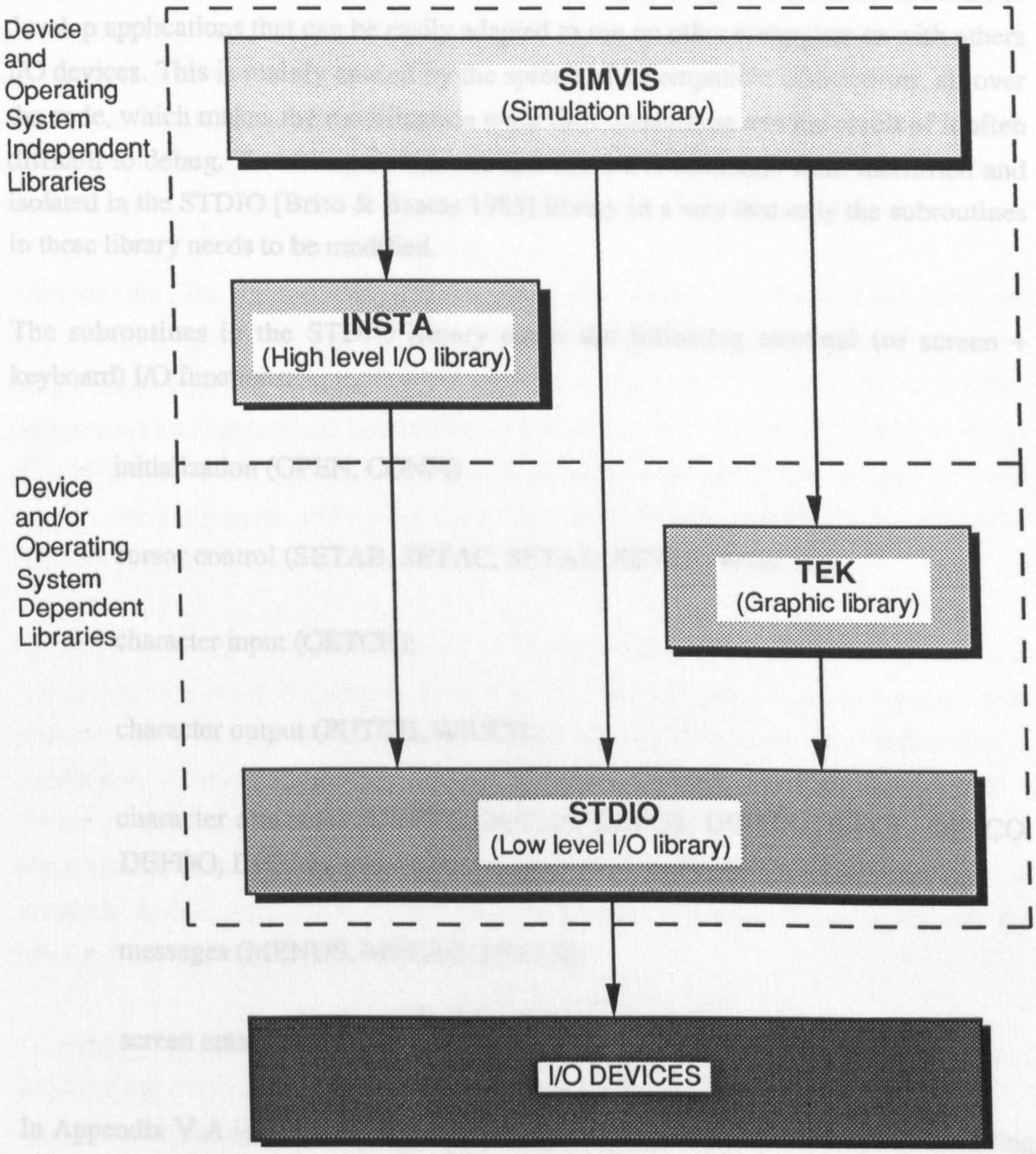


Figure 5.5 Overall organization of the SIMVIS libraries

5.3.1.2. STDIO Low Level I/O Library

The reason behind the creation of this library was the recognition of the terminal type and operating system dependence of low level I/O functions. In interactive and graphics applications it is usually necessary to have a low level control of the I/O devices for

developing a good and reliable user interface. Unfortunately this makes it difficult to develop applications that can be easily adapted to run on other computers or with others I/O devices. This is mainly caused by the spread of incompatible instructions, all over the code, which makes the modification work time consuming and the result of it often difficult to debug. To overcome this the low level I/O functions were identified and isolated in the STDIO [Brito & Bastos 1985] library in a way that only the subroutines in these library needs to be modified.

The subroutines in the STDIO library cover the following terminal (or screen + keyboard) I/O functions:

- initialization (OPEN, CONFI)
- cursor control (SETAB, SETAC, SETAD, SETAE, WRICA);
- character input (GETCH);
- character output (PUTCH, WRICH);
- character attributes (DEFBL, DEFCA, DEFCB, DEFCF, DEFCL, DEFCO, DEFDO, DEFGL, DEFUN);
- messages (MENUS, MERRO, BELLE);
- screen erase (EREOL, ERPAG, LIMPA).

In Appendix V.A is given a complete description of the STDIO subroutines including the purpose and the type and meaning of their parameters.

5.3.1.3. TEK Graphics Library

When someone is developing an application using graphics one of the first things to decide is what graphic devices are to be used. There is no world-wide accepted standard yet concerning the interface and functionality of graphic terminals. Nevertheless the Tektronix graphics terminals are well established in this field and their use is popular and supported by many applications. So Tektronix seemed the right choice for using

graphics in visual interactive simulation models developed with SIMVIS. The terminal model, the 4107/4109 was chosen because it was new at the time, and incorporated enough graphics functionality for the desired objectives at a reasonable cost. For use with those terminals several graphic input devices were available like joydisk, joystick, mouse and tablet. Also to obtain a graphics printout a plotter or screen colour hardcopy could be used.

After selecting the graphic devices, it was necessary to find a solution to interface them with the application. The better choice was probably one of the graphics standards as GKS (Graphic Kernel System: ISO (IS 7942); ANSI (ANS X3.124-1985) or PHIGS (Programmer's Hierarchical Interactive Graphics System: ISO 9592-1:1988(E); ANSI X3.144-1988). These standards are operating system and hardware independent and support several graphic I/O devices. Furthermore GKS and PHIGS can be used with different programming languages.

Although knowing all the advantages of using standards there are certain situations where their use could be compromised. One of these situations is when none of these standards are available. This can happen either because there is not an implementation for the specific environment or because there is not the driver for the desired graphics device. Another situation is when acceptable performance levels cannot be reached with those standards. As they are general purpose, not optimized for any particular field or graphics device, they can sometimes be inefficient and become inadequate for the development of an application.

In SIMVIS, after looking at various alternatives, it was decided to develop a FORTRAN 77 interface library TEK [Brito 1984] for the tektronix 4107/4109 terminals. This seemed the better solution to guarantee efficiency, portability and flexibility even if the price paid was a higher development effort. The efficiency was achieved because no overheads were introduced and all the local facilities of the terminal could be used. The portability was fair because TEK library could be transferred to other operating systems providing the same type of terminal or a compatible one was used. The flexibility exists to add new functions or to adapt the existing ones to new graphics devices. In appendix V.B is given a complete description of the TEK library subroutines.

5.3.1.4. INSTA High Level I/O Library

The configuration stage of SIMVIS data-driven generic models (see 5.2.1) requires the input of a very large amount of information. So when developing a data-driven generic model for a particular area a significant effort needs to be made regarding the screen layout, field validation and user interface. This made obvious the importance of special I/O tools that could reduce the development time, make more robust data input, improve the user interface, and make it easier to do modifications. There were the main reasons for the creation of the INSTA [Brito & Bastos 1985] library and the interactive data-entry screen generator MSC [Brito 1985].

The subroutines in the INSTA library are grouped according their function as follows:

- high level write (ESCRI, ESINC, ESINT, ESREA, ESREC);
- high level read (LECAR, LECLI, LEINT, LEREA);
- screen management (LEDEV, LEFOVC, LERMA, LERMV);
- auxiliary (LENGH).

In appendix V.C is given a complete description of the INSTA library subroutines.

5.3.1.5. SIMVIS Simulation Library

As it was stated previously (see 5.1) SIMVIS [Bastos & Moreira da Silva 1985] is an event-based visual interactive general purpose system developed in FORTRAN 77. Since the beginning of its development, some changes and upgrades were made reflecting the needs of specific models. Improvements were made on the simulation library overall efficiency. Related to this, perhaps the most significant change, was the addition of new subroutines where the simulation elements were identified by a pointer instead of a name. The following example of an entity attribute definition shows the difference:

```
Call DANENT ('TRUCK', ...)          using the entity name;
```

Call DANENP (IPOENT, ...) using the entity pointer;

although the second method is less obvious it is however more efficient. This occurs because with the pointer the entity attribute can be addressed directly and with the name there is an overhead of work to find the corresponding pointer. Still the user can decide where it is opportune to use one method or the other.

In the AWARD system some of the SIMVIS simulation library functionality is used. Facilities like automatic data collection for 'time in the queue' or 'queue length', histograms and other simulation elements were not used. In appendix V.D is given a description of the most important SIMVIS simulation library subroutines used in AWARD that were grouped by function as follows:

- auxiliary (EDELAY, INFOZ0, NOMPOI, VERNOM, WRITZ0);
- dump management (DUMP00, DUMPII, REST00, RESTII);
- entity management (CRIENT, DALENP, DANENP, LALENP, LANENP, LCANEN, READAT);
- error management (SERROR);
- event management (ADENEV, CRCLOC, IDENTI, NEXTEN, SCHEDU, SYSEVE, TAENEV, UNSCHE, USEEVE);
- initialization (INITZ0);
- interactions (INTSYS);
- logical display management (CRIECR, DANECR, INFECR, LANECR, LCAECR, READIS, SCREOF, SCREON);
- queue management (ADDFIR, ADDLAS, ADSQUE, CRIQUE, IDEQUE, ISIQUE, TAKENT, TAKFIR, TAKLAS, TAKPOS);
- time management (ADDTIME, NSTIME);

- vector management (CREVEC).

5.3.1.6. The interactive data-entry screen generator MSC

The MSC [Brito 1985] utility was developed because of the necessity of quickly creating and editing data-entry screens. It is an interactive application that has facilities to define the screen layout and create different types of data fields (e.g. real, integer or character). All this information can be saved to a file that can be retrieved later for modifications or used by INSTA subroutines, in a user application, for data screen management (see figure 5.6).

After starting the MSC utility two options are available: create a new data screen; or edit an existing one. If the first option is selected it is then necessary to define if the terminal screen is to be cleared before displaying the data and what colour should be used. Afterwards the user can move the cursor, with the arrow keys, around the screen or choose one of the options that are displayed at the bottom. These are selected by pressing a letter on the keyboard and performing the following functions:

(N) field creation: creates a data field for display and read data within an application using the INSTA subroutines (see 5.3.1.4);

(T) text: creates text on the screen;

(A) erase: deletes a data or text field on the screen;

(Esc) end: quits the utility and let the user save the screen data to a file;

(P) position: displays at the bottom right of the screen the line and column coordinates for the current cursor position;

(R) repeat: repeats a data or text field on the screen;

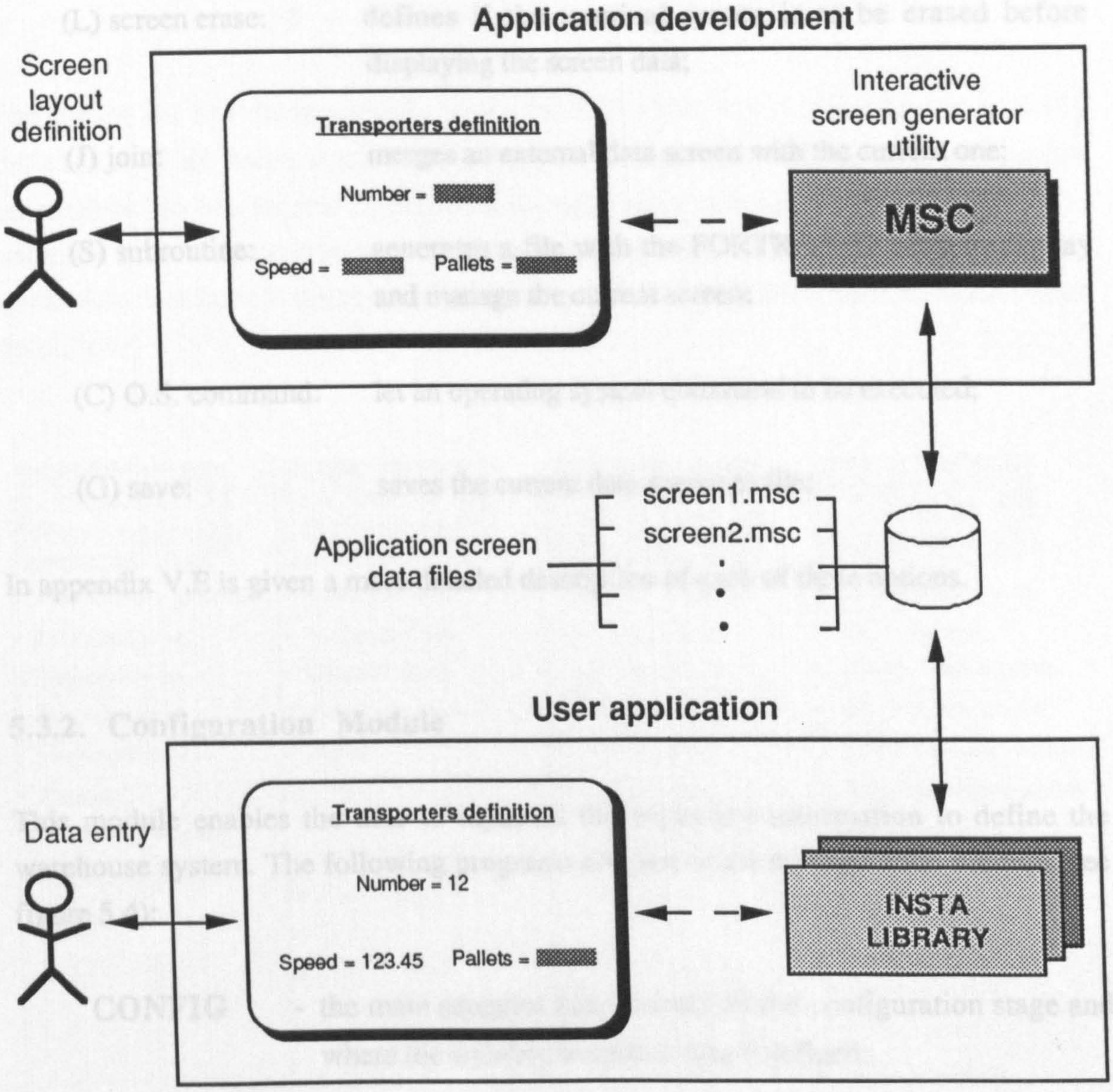


Figure 5.6 The MSC interactive screen generator environment

(V) variable repeat: repeats a data field during the running of the user application;

(D) move: moves a data or text field around the screen;

(E) edit: edits a data or text field;

(I) information: gives full information for each of the data fields;

(L) screen erase: defines if the terminal screen is to be erased before displaying the screen data;

(J) join: merges an external data screen with the current one;

(S) subroutine: generates a file with the FORTRAN 77 code to display and manage the current screen;

(C) O.S. command: let an operating system command to be executed;

(G) save: saves the current data screen to file;

In appendix V.E is given a more detailed description of each of these options.

5.3.2. Configuration Module

This module enables the user to input all the necessary information to define the warehouse system. The following programs are part of the configuration module (see figure 5.4):

CONFIG - the main program that controls all the configuration stage and where the warehouse system data is defined;

PG - an auxiliary program to create or modify product data files;

OUTORDER - an auxiliary program that generates outorder files based on statistical data.

CONFIG is a menu-driven program where most of the data is input graphically using CAD (Computer Aided Design) techniques. After the initial password screen the following menu is displayed:

1 - Define physical parameters
2 - Define working parameters
3 - Define the product group file
4 - Define the outorders file

5 - Create the initial running data file

The reason for having separated options for data input was a practical one. Usually there is a need for testing a particular warehouse configuration with different operational parameters. So it is logical to organize the data input in a way that makes it easy to combine several alternatives in order to evaluate them. Figure 5.7 shows how the warehouse data input is organized and the major topics involved in the simulation model definition.

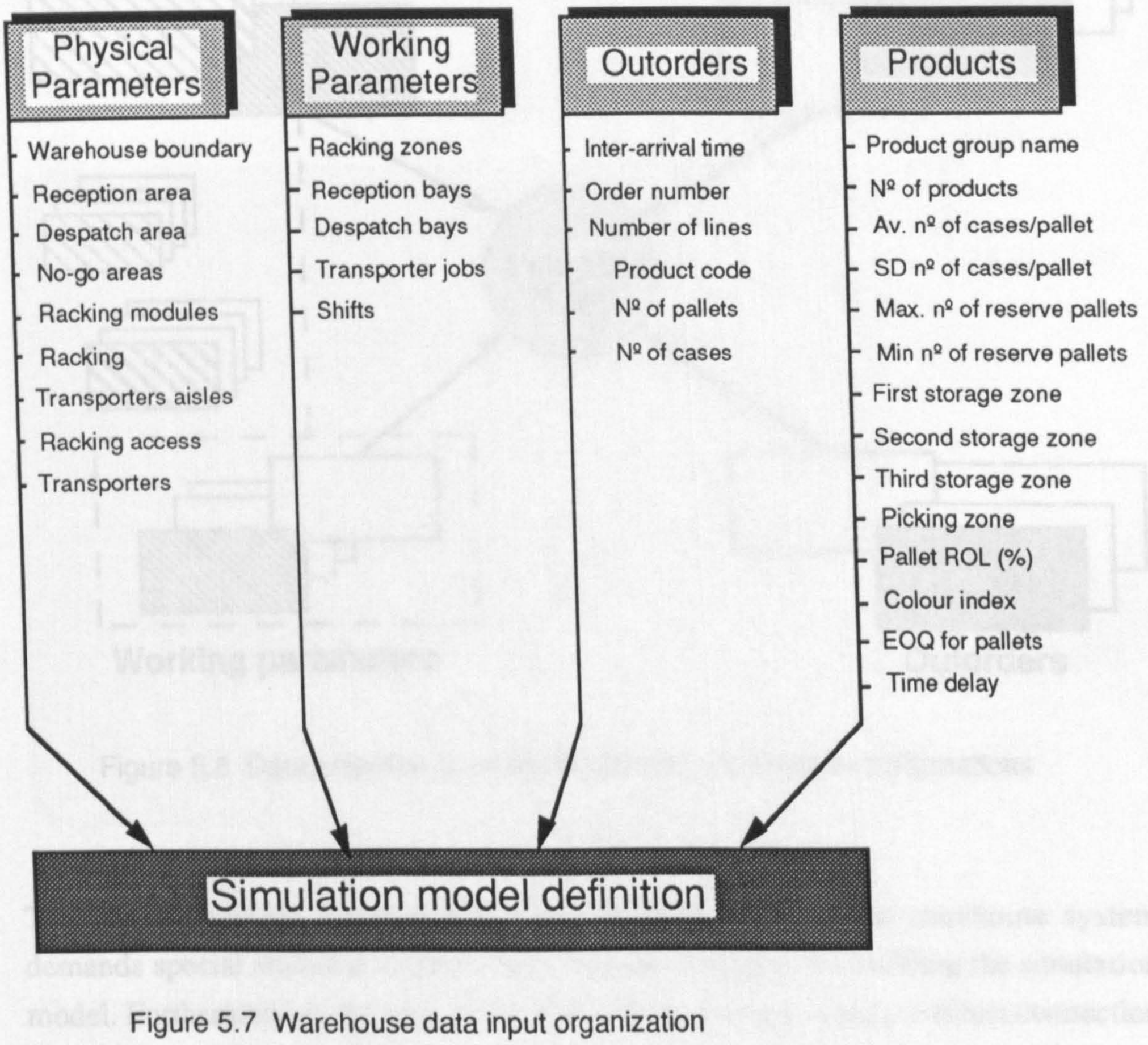


Figure 5.7 Warehouse data input organization

The use of this approach gives the required flexibility to address a large number of situations. If a new warehouse is going to be built in a green field situation, probably the design team will want to evaluate different physical layouts and technological solutions combined with the expected throughput and product range variety. In a case like this the outorders and product information will be kept constant and used jointly

with the various sets of physical and working parameters to build the simulation model for each configuration. However, if a warehouse layout is already defined, as in an existing warehouse, then it will be possible for the same physical parameters to test different working parameters, outorders policies or product ranges. Figure 5.8 shows the flexibility in data selection for evaluating different warehouse configurations.

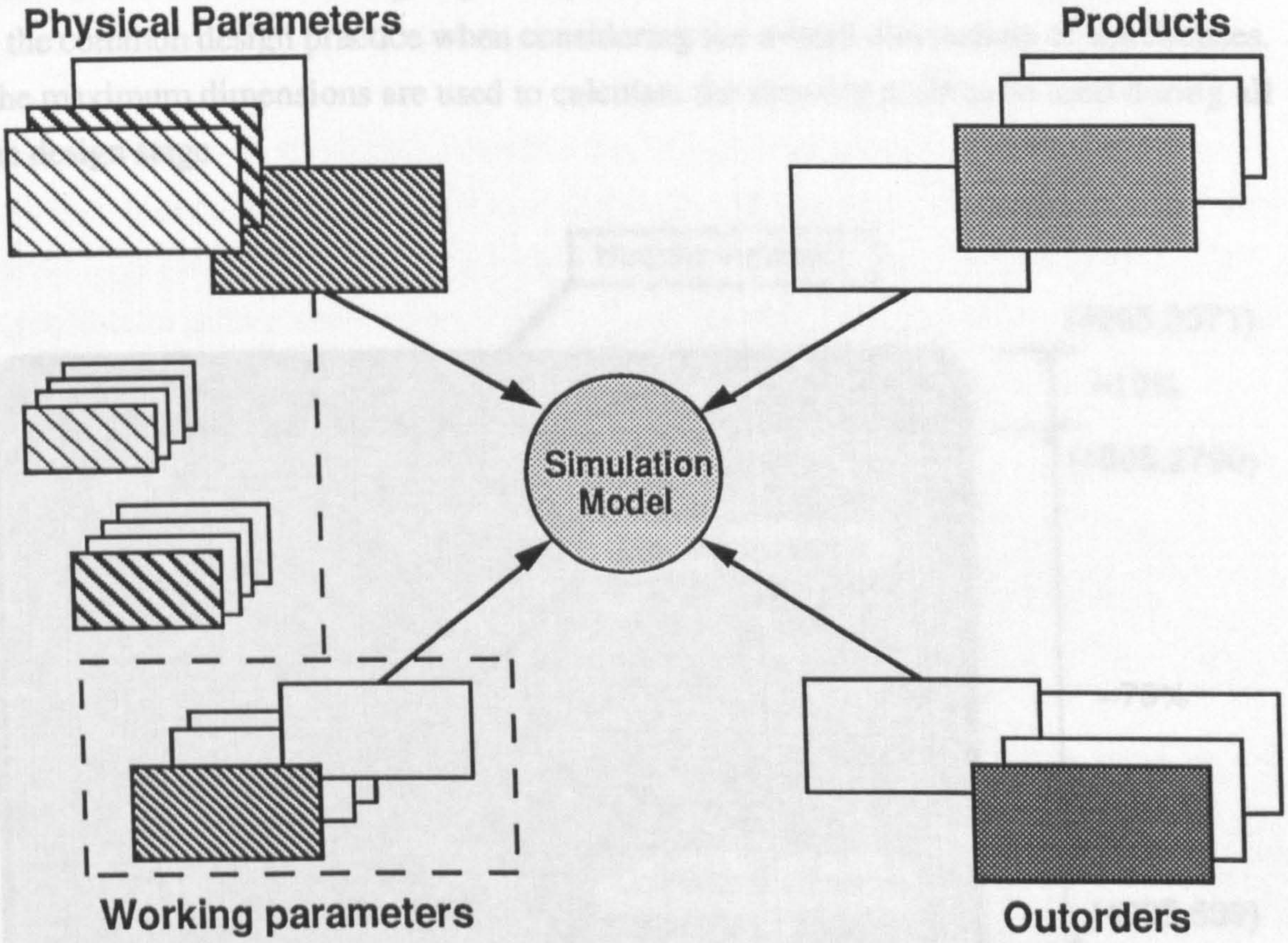


Figure 5.8 Data selection to evaluate different warehouse configurations

The use of multiple interdependent data merged to define the warehouse system demands special attention to ensure a correct environment for building the simulation model. Furthermore, in the case of the working parameters, there is a direct connection between them and the physical parameters (see figure 5.8). This means that the data definition under the working parameters (see figure 5.7) must be done according to a specific physical configuration previously defined.

5.3.2.1. Physical Parameters Definition

In this option is defined the warehouse physical layout and the number of transporters and their characteristics. Before starting to enter the warehouse data it is necessary to set the drawing area. This is made by asking the user the maximum width and length of the area where the warehouse is going to be placed. All the values are entered in metres as it is the common design practice when considering the overall dimensions of warehouses. The maximum dimensions are used to calculate the drawing scale to be used during all the design stage.

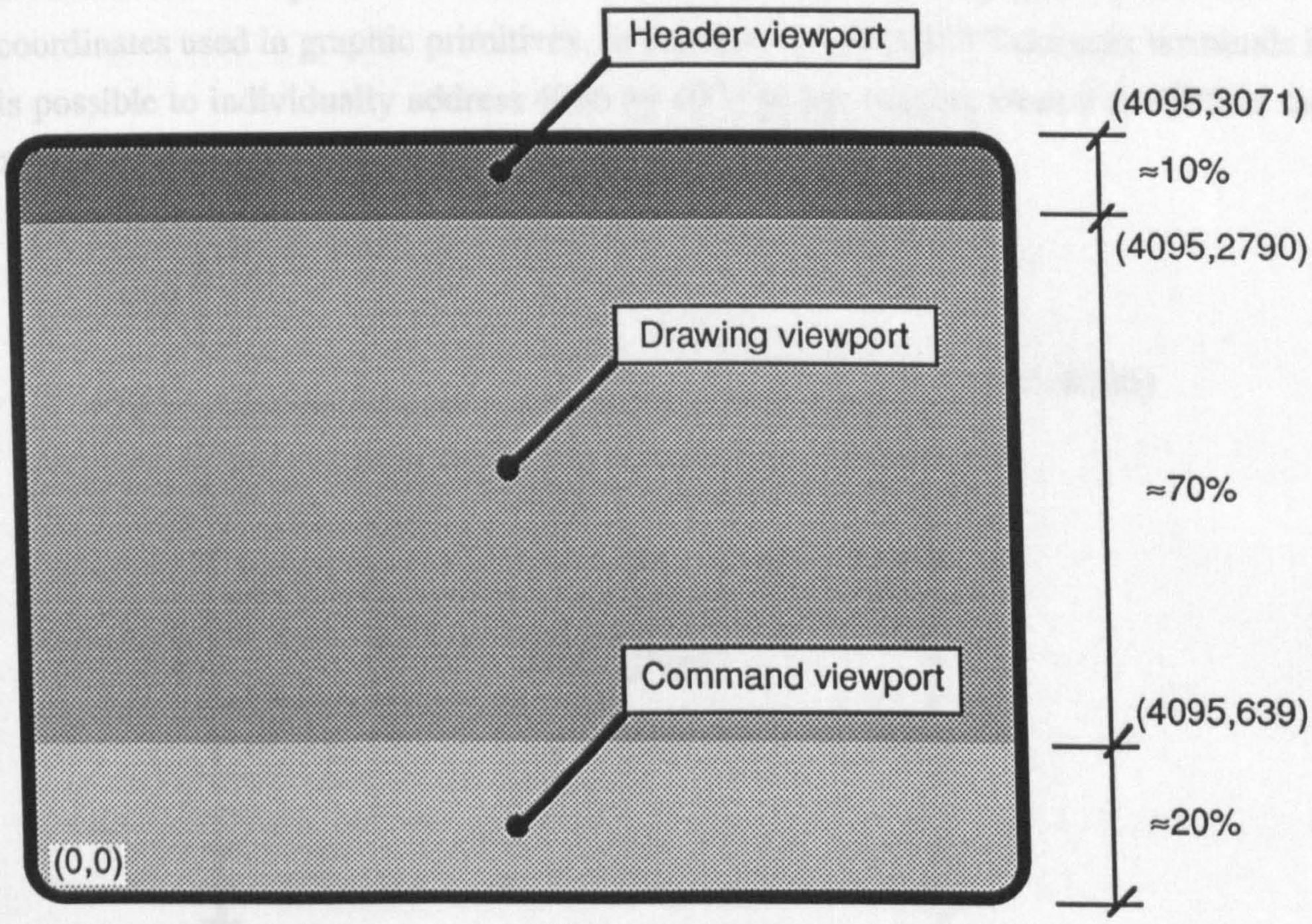


Figure 5.9 Graphic device screen viewports

The available graphics device screen area is divided in to the following three parts (see figure 5.9): the header viewport, the drawing viewport, and the command viewport. A viewport is a specific physical area of a graphic device where graphical information is displayed. The header viewport is used to display the module header and the current application name. During the simulation the real time and date are also displayed in this viewport. In the drawing viewport during the configuration stage, the definition and display of the warehouse layout is made. During simulation the graphic animation of the

model appears here. In the configuration module the menus and additional information are presented in the command viewport. In the simulation module, this viewport contains the interaction box with the command names and the user dialogue area.

In the Tektronix 4107/4109 terminals the viewport corners are specified in normalized screen coordinates. Independently of the physical screen size and resolution the width is mapped into values ranging between 0 and 4095 and the height between 0 and 3071. Figure 5.9 shows the upper right coordinates of each viewport.

When working with graphics it is common practice to use integer coordinates defined in a virtual address space. This virtual space sets the lower and upper limits for the coordinates used in graphic primitives. In the case of 4107/4109 Tektronix terminals it is possible to individually address 4096 by 4096 points ranging from 0 to 4095 in the xx and yy directions (see figure 5.10).

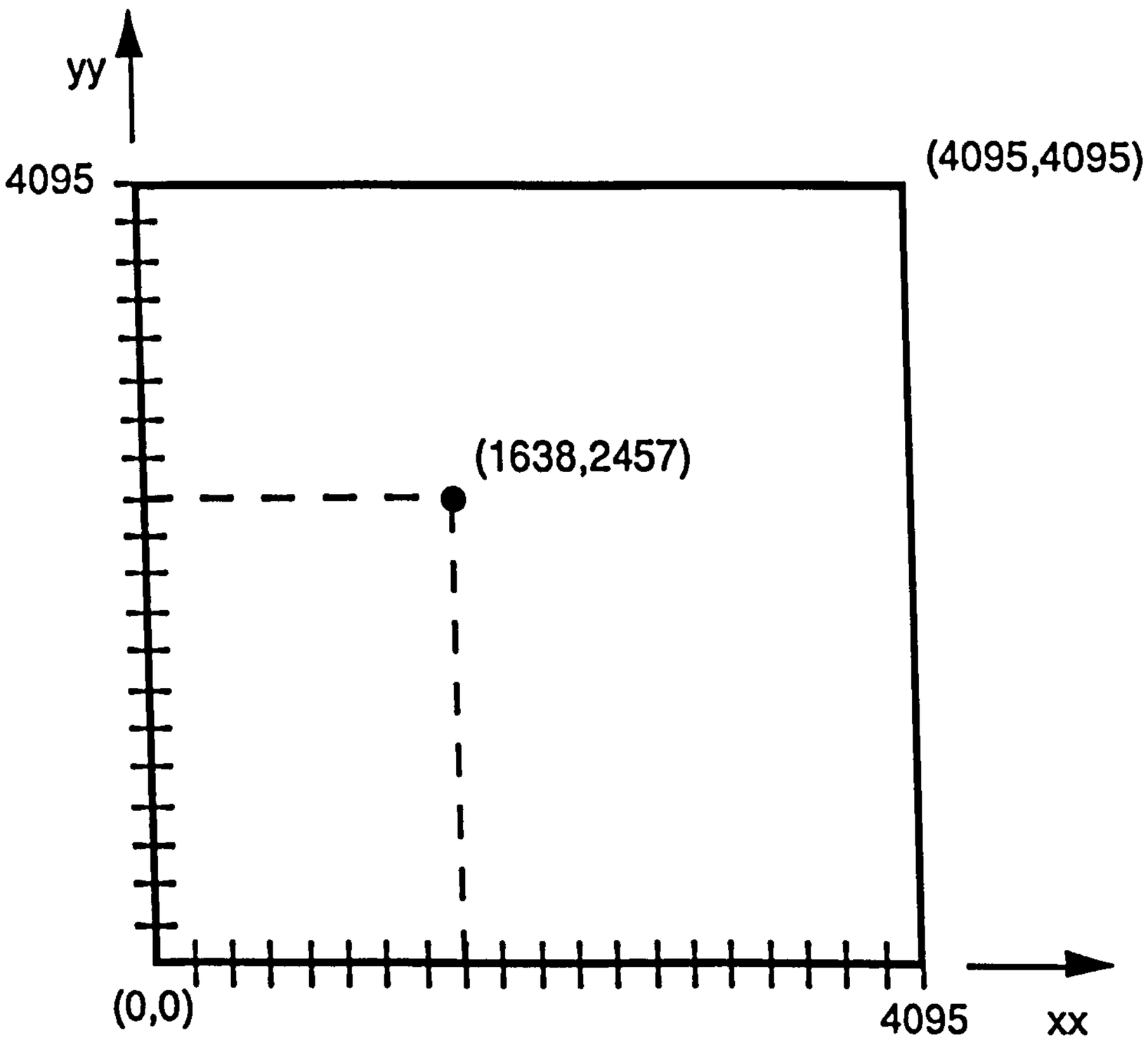


Figure 5.10 Virtual address space in tektronix 4107/4109 terminals

The graphics in the virtual space are always displayed on the screen in an area associated with a viewport. It is also possible to display only a part of the virtual space.

This introduces the important concept of a graphic window that is the area of the virtual space that is visible in the viewport. The window and viewport concepts involve a series of coordinate transformations whose purpose is to map part of the virtual space in the window frame into the device coordinates in the viewport frame (see figure 5.11). Although in figure 5.11 the picture appearing in the viewport is proportional to that corresponding in the window of the virtual space, that is not always true. That depends on the window, viewport and screen coordinates whose values are entered in the various transformations carried out.

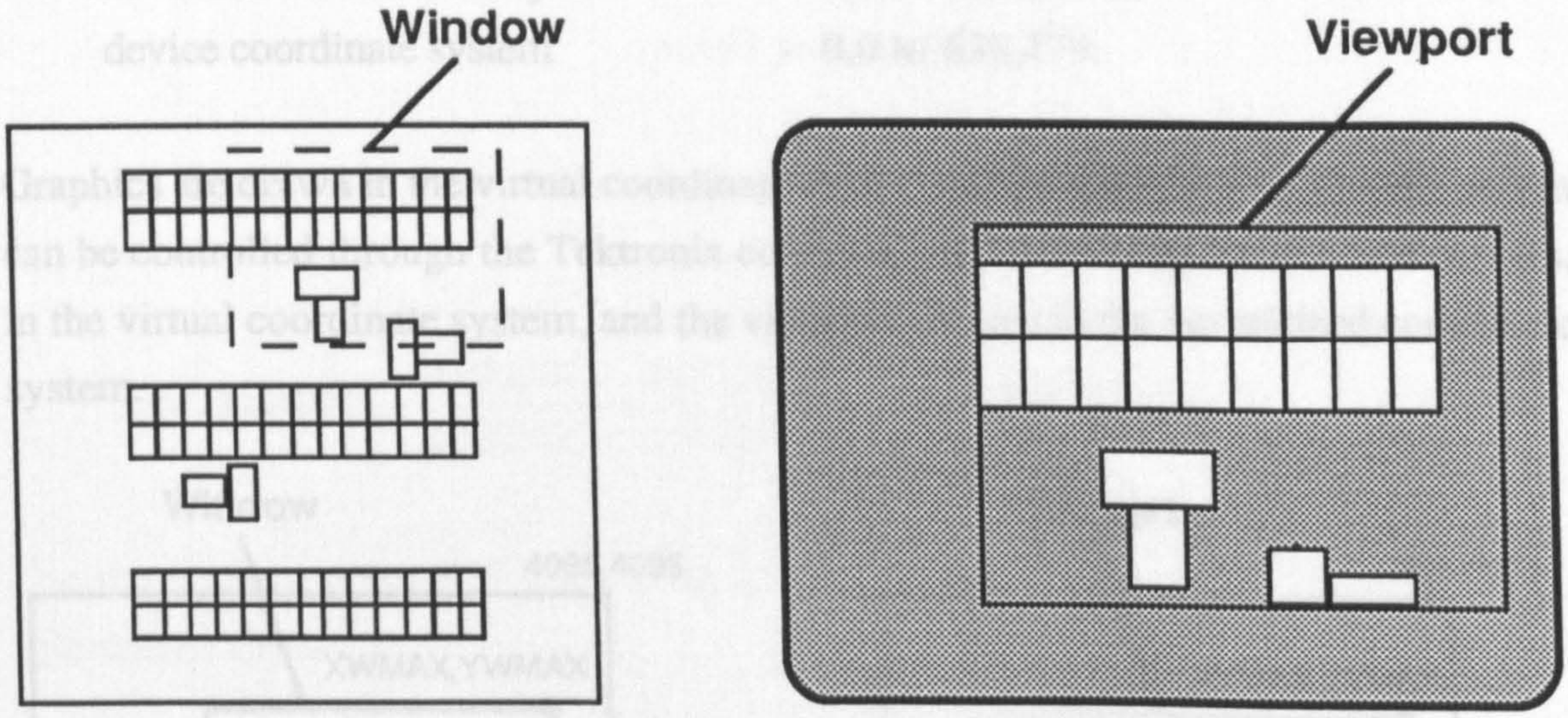


Figure 5.11 The window and viewport graphic concepts

Before continuing it is important to clarify the meaning of the following coordinate systems:

virtual coordinate system - it is an integer coordinate system, independent of the graphic device, that is used directly to draw graphic primitives;

normalized coordinate system - it is an integer coordinate system, independent of the graphic device, that is used to specify the graphic device area where the graphics will be displayed;

device coordinate system - it is an integer coordinate system corresponding to the graphic device resolution in dots or pixels (picture elements).

For the Tektronix 4107/4109 terminals the range for the different coordinate systems is:

virtual coordinate system - 0,0 to 4095,4095;
normalized coordinate system - 0,0 to 4095,3071;
device coordinate system - 0,0 to 639,479.

Graphics are drawn in the virtual coordinate system and the way they look on the screen can be controlled through the Tektronix command set by defining the window corners, in the virtual coordinate system, and the viewport corners in the normalized coordinate system.

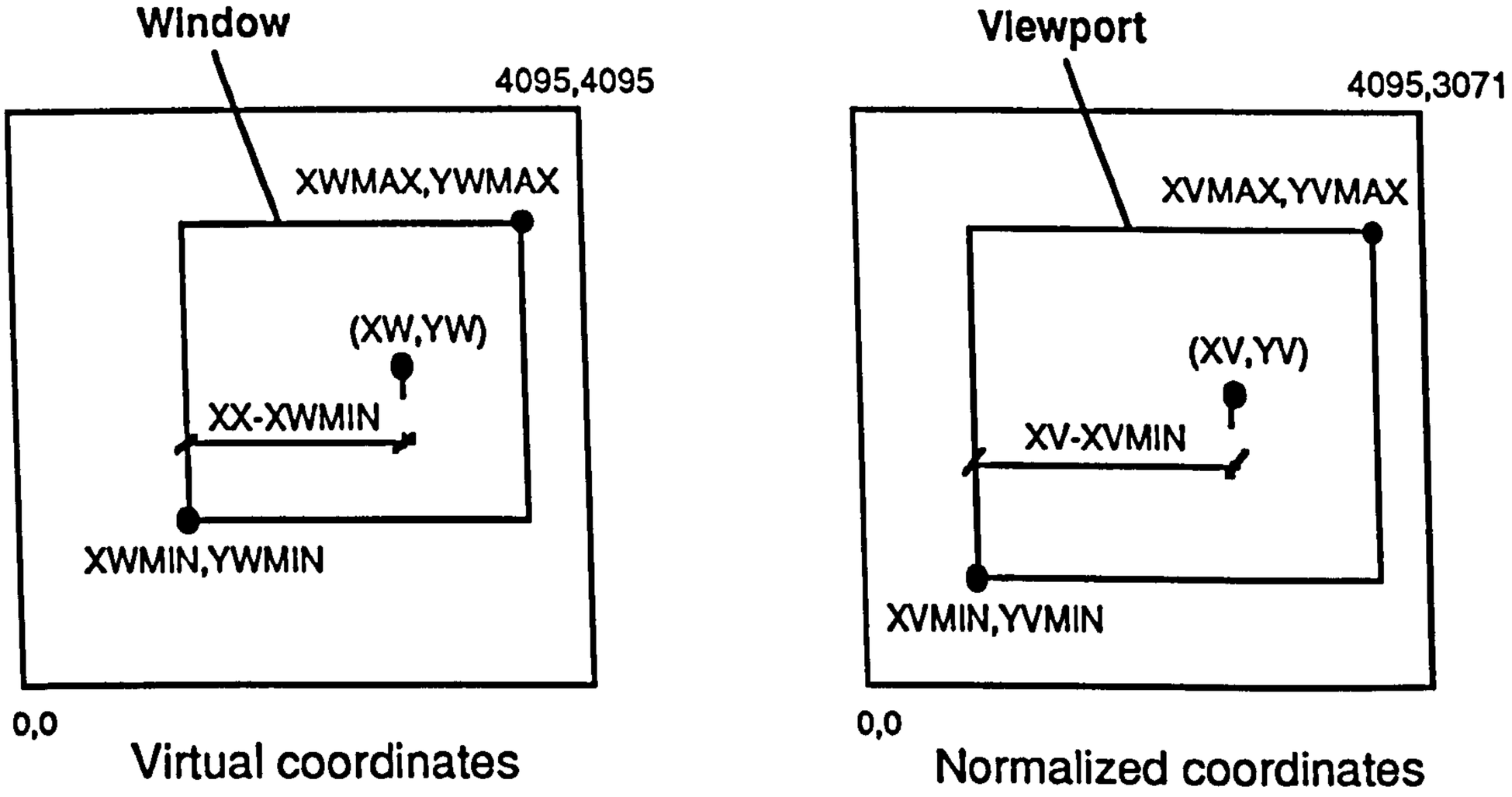


Figure 5.12 The mapping of virtual coordinates into normalized coordinates

The transformation between virtual coordinates and normalized coordinates is achieved by mapping the window area, in the virtual coordinate system, into the viewport area in the normalized system (see figure 5.12). So, if we concentrate only in the XX direction we obtain the following proportion:

$$\frac{XW - XWMIN}{XWMAX - XWMIN} = \frac{XV - XVMIN}{XVMAX - XVMIN} \qquad (5.1)$$

where XWMIN,XWMAX are the window XX extreme coordinates;
XVMIN,XVMAX are the viewport XX extreme coordinates;
XW is the XX virtual coordinate of a general point inside the window;
XV is the XX normalized coordinate of a general point inside the viewport.

From (5.1) we obtain the value of the general point XX normalized coordinate as follows:

$$XV = XVMIN + \frac{XVMAX - XVMIN}{XWMAX - XWMIN} \times (XW - XWMIN) \qquad (5.2)$$

For the generic point YY normalized coordinate is obtained, in a similar way, the following expression:

$$YV = YVMIN + \frac{YVMAX - YVMIN}{YWMAX - YWMIN} \times (YW - YWMIN) \qquad (5.3)$$

where YWMIN, YWMAX are the window YY extreme coordinates;
YVMIN, YVMAX are the viewport YY extreme coordinates;
YW is the YY virtual coordinate of a general point inside the window;
YV is the YY normalized coordinate of a general point inside the viewport.

The advantage of the normalized coordinate system is that it is device independent and from which it is easy to do the necessary transformations for different graphic devices. Furthermore its coordinates are usually in the range between 0 and 1 to reduce the amount of calculations involved with those transformations. As it was said before this is not true in the Tektronix 4107/4109 graphic terminals (see figure 5.12).

The final transformation between the normalized coordinates and the device coordinates is easily done considering that the normalized space is mapped into the entire graphic device area. If in the two systems the minimum values for the coordinates in both directions is 0 then we have the following proportion:

$$\frac{XNMAX}{XV} = \frac{NUPIXX}{XD} \qquad (5.4)$$

where XNMAX is the maximum XX normalized coordinate;
NUPIXX is the maximum XX graphic device coordinate;
XD is the general point XX graphic device coordinate.

From (5.4) it is possible to obtain the final value for the XX graphic device coordinate as follows:

$$XD = \frac{NUPIXX}{XNMAX} \times XV \qquad (5.5)$$

In the same way we would obtain the YY graphic device coordinate:

$$YD = \frac{NUPIXY}{YNMAX} \times YV \qquad (5.6)$$

where YNMAX is the maximum YY normalized coordinate;
NUPIXY is the maximum YY graphic device coordinate;
YD is the general point YY graphic device coordinate.

Using the value of XV and YV from expressions (5.2) and (5.3) in (5.5) and (5.6) we relate the device coordinates with the virtual coordinates obtaining the following expressions:

$$XD = INT\left(\frac{NUPIXX}{XNMAX} \times \left(XVMIN + \frac{XVMAX - XVMIN}{XWMAX - XWMIN} \times (XW - XWMIN)\right) + 0.5\right) \qquad (5.7)$$

$$YD = INT(\frac{NUPIXY}{YNMAX} x (YVMIN + \frac{YVMAX - YVMIN}{YWMAX - YWMIN} x (YW - YWMIN)) + 0.5) \quad (5.8)$$

Because the device coordinates are usually integers (dots or pixels) we add 0.5 for rounding purposes and take the integer part. In the case of the Tektronix 4107/4109 terminals those transformations are handled directly by their command set.

Although the concept of virtual coordinates is good enough in most graphics applications, it can be insufficient in other applications when real value coordinates are required. When this happens it is necessary to introduce a new coordinate system which is normally called the world coordinate system. With this concept it is possible to work with graphics using the same units as in the real world. Unfortunately the graphic device command set will support at most, the virtual coordinate system as happens with the Tektronix terminals.

The graphics packages or libraries like the standards GKS and PHIGS (see 5.3.1.3.) work with world coordinates. If the graphics environment does not have these facilities then the expressions (5.7) and (5.8) can be used with XW and YW being world coordinates instead of virtual coordinates. Another solution is to deduce the expressions to transform world coordinates into virtual coordinates. Because the entire world space is mapped into the virtual space these expressions will be identical to expressions (5.2) and (5.3).

After having introduced some essential graphics concepts, we return to the initial problem of setting the drawing area and graphics scale to enable the warehouse definition. The part of the screen used for this purpose is what was called the drawing viewport and was defined in figure 5.9.

The Tektronix virtual address space used was as large as possible without having graphic distortion. The distortion appears when the ratio between the window width and length is different from the ratio of the corresponding viewport width and length. As the maximum value along XX for the virtual coordinates is equal to the value of the viewport size in the same direction (see figure 5.13) so, to maintain the same ratio, the maximum value for the virtual coordinates along YY must be equal to the viewport size in the same direction.

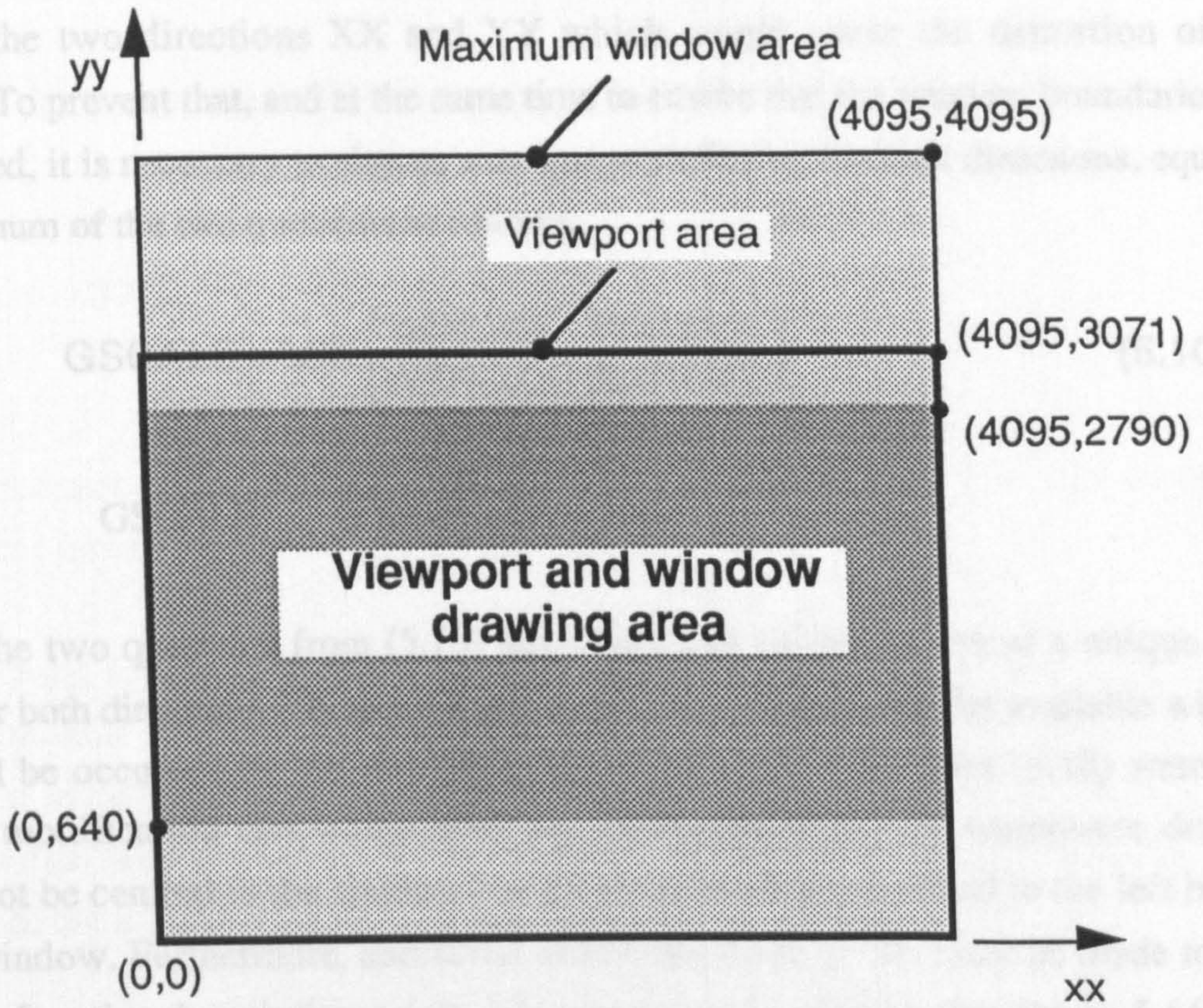


Figure 5.13 Viewport and window drawing area

Because the window and viewport have the same dimensions it was also decided, to simplify the calculations, to make the window origin coordinates coincide with the drawing viewport bottom left coordinates (see figure 5.13). For mapping the warehouse area (in square meters) to the drawing area the same process used in (5.4) will result in the following expressions:

$$XV = \frac{WINDXX}{WARDXX} \times XX \quad ; \quad YV = \frac{WINDYY}{WARDYY} \times YY \qquad (5.9)$$

where WINDXX is the drawing window XX dimension;
WINDYY is the drawing window YY dimension;
WARDXX is the maximum warehouse XX dimension;
WARDYY is the maximum warehouse YY dimension;
XX, YY are a warehouse general point coordinates (m);
XV, YV are a window general point virtual coordinates.

If the quotients in the two expressions (5.9) are not equal, that means a different graphic scale in the two directions XX and YY which would cause the distortion of the drawing. To prevent that, and at the same time to ensure that the window boundaries are not crossed, it is necessary to choose a unique scale factor, for both directions, equal to the minimum of the two quotients as follows:

$$GSCALE = MIN\left(\frac{WINDXX}{WARDXX}, \frac{WINDYY}{WARDYY}\right) \qquad (5.10)$$

where GSCALE is the graphic scale factor.

Unless the two quotients from (5.10) have the same value, the use of a unique scale factor for both directions XX and YY will mean that only a part of the available window area will be occupied by the warehouse drawing. If the expression (5.10) were used without modification this would have the consequence that the warehouse drawing would not be centred in the window but it would be always justified to the left bottom of the window. Furthermore, additional modifications to (5.10) must be made to cope with the fact that the window origin YY coordinate is not zero (see figure 5.14). The expressions (5.10) need to be modified because it was assumed that the window origin coordinates were (0,0). To change the expressions (5.10) it is only necessary to add the new origin coordinate values. These values can be obtained from the following expressions:

$$NCORIX = OFFSXX + \left(\frac{WINDXX - INT(WARDXX \times GSCALE + 0.5)}{2}\right) \qquad (5.11)$$

$$NCORIY = OFFSYY + \left(\frac{WINDYY - INT(WARDYY \times GSCALE + 0.5)}{2}\right) \qquad (5.12)$$

where NCORIX is the new window XX origin coordinate;
OFFSXX is the XX offset of the initial window;
NCORIY is the new window YY origin coordinate;
OFFSYY is the YY offset of the initial window.

Now the final version of expressions (5.9) will be:

$$XD = NCORIX + INT(XX \times GSCALE + 0.5) \quad (5.13)$$

$$YD = NCORIY + INT(YY \times GSCALE + 0.5) \quad (5.14)$$

where NCORIX is obtained from expression (5.11);
NCORIY is obtained from expression (5.12);
GSCALE is obtained from expression (5.10).

These expressions are used through the configuration stage to convert warehouse coordinates (in metres) to the Tektronix virtual coordinates.

After having set the drawing area and graphics scale it is possible to define the warehouse data. When creating a new configuration it is necessary to follow a certain data input sequence, but when editing an existing configuration the data can be entered in any order. The physical data definition is made in different sections each one corresponding to options of the following menu:

1 - Define the warehouse boundary
2 - Define the reception area
3 - Define the despatch area
4 - Define the no-go areas
5 - Define the racking modules
6 - Define the racking
7 - Define the aisles
8 - Define the racking access
9 - Define the transporters

Most of the data input is made graphically using a joydisk or the keyboard arrows. A top view from the warehouse is used for the 2D drawing. A bar menu at the bottom of the screen gives, in each option, access to a set of commands that are selected with the arrow keys and are executed by pressing the enter key. Next is given a complete description of several options under the physical layout definition.

5.3.2.1.1. Warehouse boundary

The warehouse boundary is defined by a series of consecutive line segments that form in the end a closed polygon. Figure 5.14 shows the screen layout during the warehouse boundary definition. At the top of the screen there is the header, in the middle the drawing area and at the bottom the command area. The bottom line of the command area is used for displaying comments or sub-menu options. The top part of the command area is used for displaying additional information.

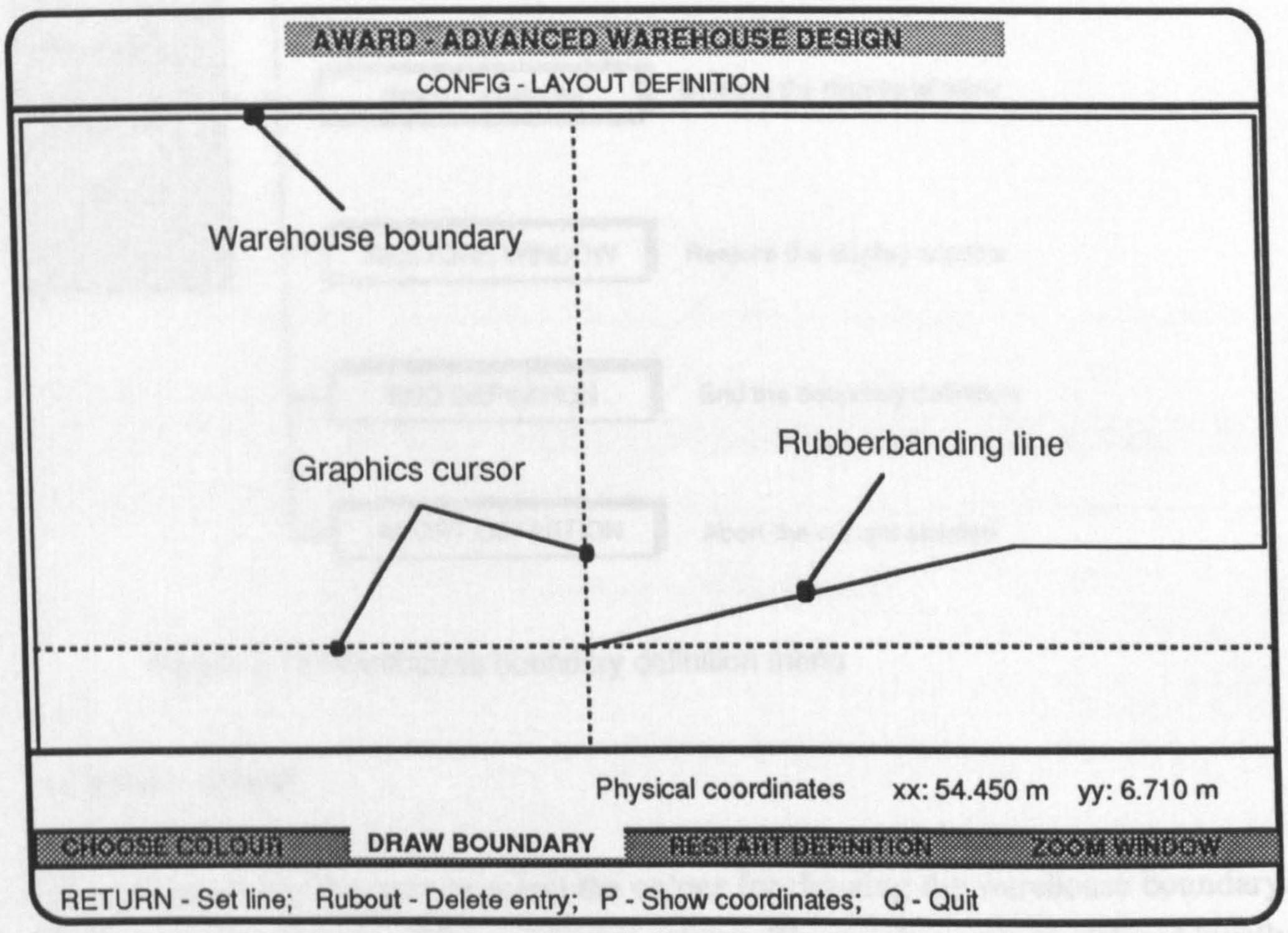


Figure 5.14 Warehouse boundary definition screen layout

The available options displayed in the bar menu are selected using the left and right arrow keys, and are executed by pressing the return key. Normally there is not enough space on the screen to display all the option names at the same time. When one of the extreme left or right options are selected and the left or right arrow keys are pressed then the bar menu is automatically updated showing more option names. Figure 5.15 presents the warehouse boundary definition menu options and the corresponding

comments that are displayed at the bottom line of the command area. Next each of this options is described.

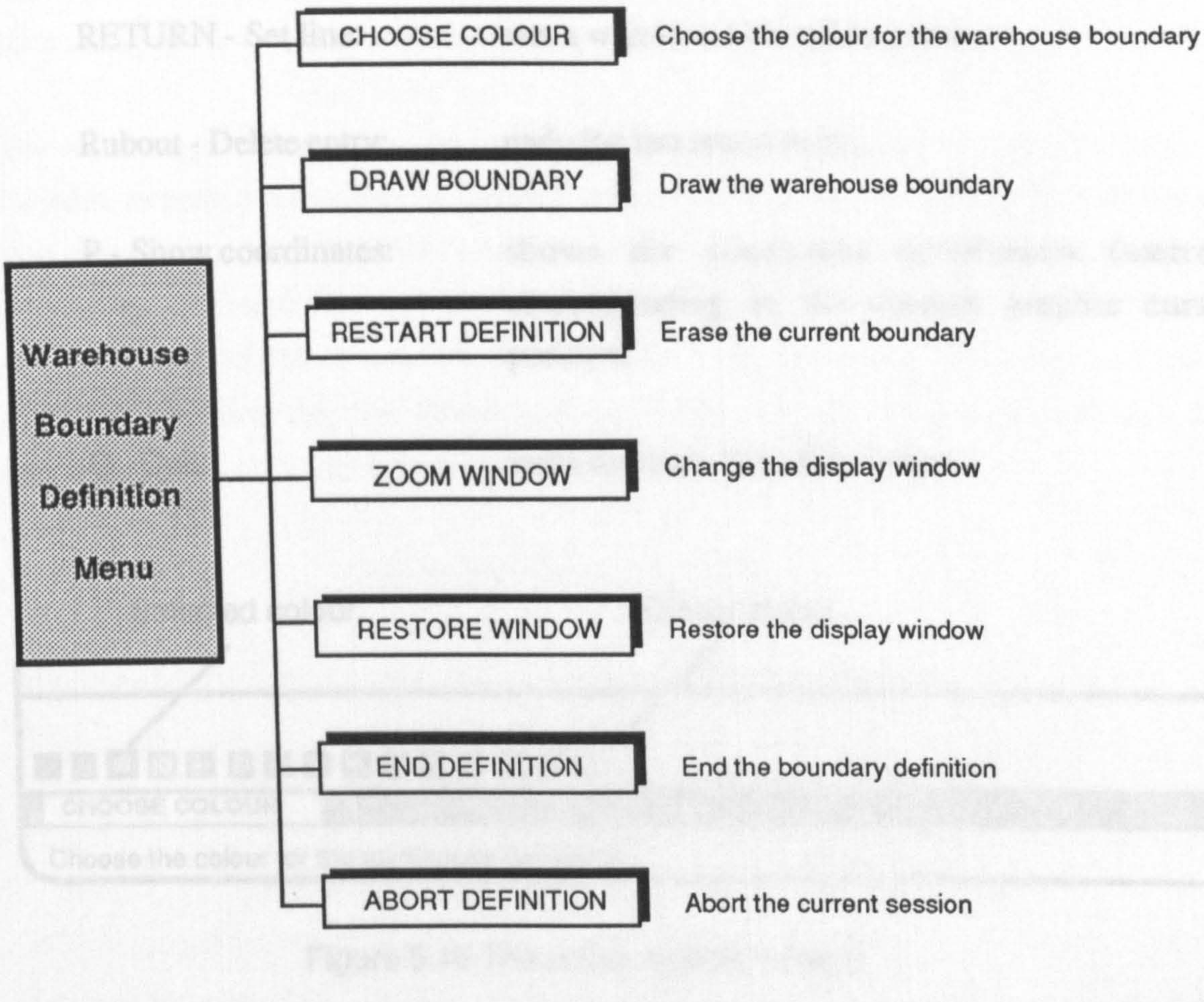


Figure 5.15 Warehouse boundary definition menu

. Choose colour

This option allows the user to select the colour for drawing the warehouse boundary. There is a choice between fifteen different colours. These colours can be defined locally on the Tektronix terminals from a pallet offering 4096 combinations. The left and right arrow keys are used to highlight the desired colour box (see figure 5.16) and then pressing the return key enables the selection and quits.

. Draw boundary

After selecting the draw boundary option the graphic cursor can be moved around the drawing area and the warehouse boundary can be created using the keys described in

the sub-menu at the bottom line of the command area (see figure 5.14). The following keys are available in the sub-menu:

RETURN - Set line: sets a warehouse boundary corner;

Rubout - Delete entry: undo the last return entry;

P - Show coordinates: shows the warehouse coordinates (metres) corresponding to the current graphic cursor position;

Q - Quit: quits the draw boundary option.

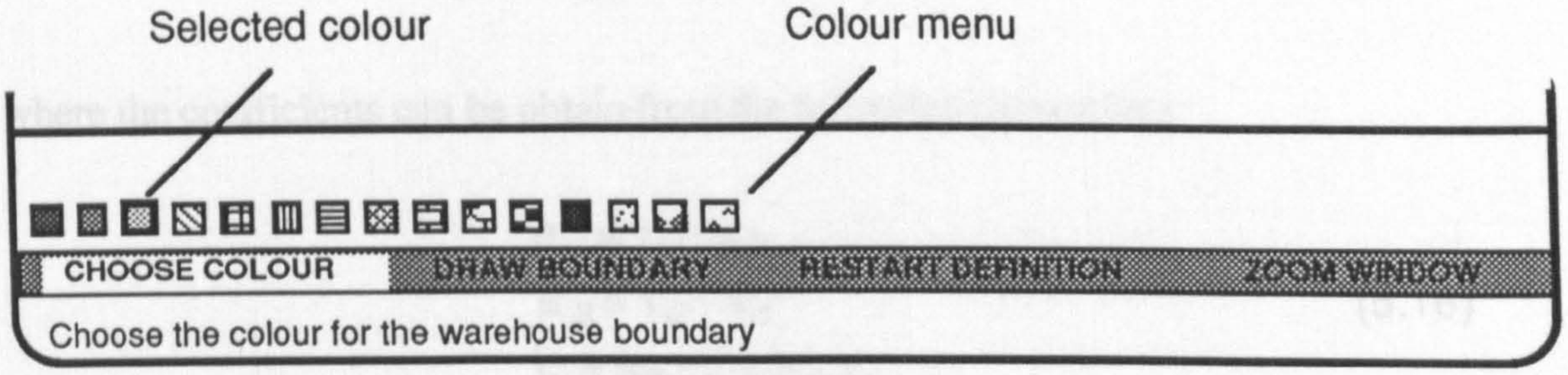


Figure 5.16 The colour selection menu

The first step in drawing the warehouse boundary is the setting of the initial corner. The graphic cursor can be moved freely and the current position coordinates can be known by pressing the 'P' key (see figure 5.14). After having positioned the cursor at the desired point for beginning the boundary, pressing the 'return' key will set that point as the initial boundary corner. Afterwards a rubberband line will always connect the last corner fixed with the current cursor position (see figure 5.14). This helps to see how the boundary line looks before setting another boundary corner. If by mistake an incorrect corner is defined, by pressing the 'return' key, then the 'rubout' key can be used to undo the action caused by that key press.

Also, a number of successive boundary corners can be removed by pressing the 'rubout' key a equivalent number of times. Pressing the 'Q' key will return the control to the bar menu level where it is possible to execute any of the available options. If the

draw boundary option is executed again, the work is restarted at the point where it was left before pressing the 'Q' key. If the warehouse boundary is not closed when leaving the warehouse boundary definition that will be done automatically. This is made by connecting the last corner with the initial corner of the warehouse boundary.

The warehouse boundary can be a closed polygon of any shape providing that none of its sides intersect each other. Whenever an attempt is made to create a new boundary line there is a check for intersections with the previously created lines and if an intersection is found then the line cannot be created. The xx and yy boundary corner coordinates are stored in two vectors along with an updated value of the current number of corners. Taking any two consecutive boundary corners with coordinates (x_{i1},y_{i1}) and (x_{i2},y_{i2}) the cartesian equation for the line i defined by them will be:

$$a_{i1}x + a_{i2}y = c_i \tag{5.15}$$

where the coefficients can be obtain from the following expressions:

$$\begin{aligned} a_{i1} &= y_{i1} - y_{i2} \\ a_{i2} &= x_{i2} - x_{i1} \\ c_i &= a_{i2}y_{i1} + a_{i2}x_{i1} \end{aligned} \tag{5.16}$$

If a new boundary line, say line j, is to be created, then a search for intersections has to be carried out in turn with each existing boundary line, say line k. To find if lines j and k intersect each other one method is to solve the simultaneous equation formed by their cartesian equations. Making i=j and i=k these equations can be obtained from (5.15) substituting in (5.16) the boundary corners coordinates that define the two lines (see figure 5.17). The result is the following simultaneous equations which solution, if exists, is the (x_i, y_i) coordinates of the intersection point:

$$\begin{aligned} a_{j1}x + a_{j2}y &= c_j \\ a_{k1}x + a_{k2}y &= c_k \end{aligned} \tag{5.17}$$

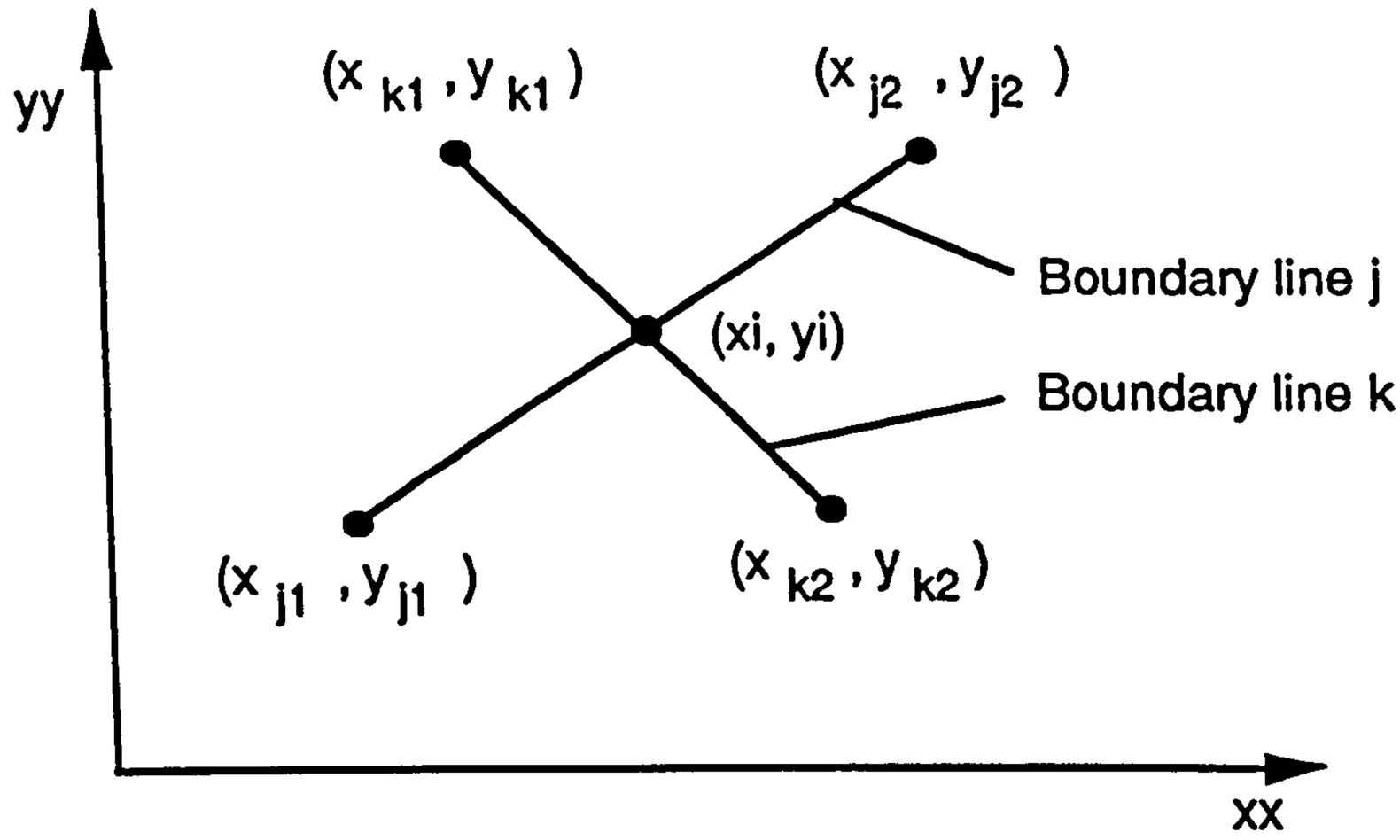


Figure 5.17 Parameters used in the checking for boundary line intersections

The coefficients matrix of the (5.17) simultaneous equations is:

$$[A] = \begin{bmatrix} a_{j1} & a_{j2} \\ a_{k1} & a_{k2} \end{bmatrix} \tag{5.18}$$

and the independent coefficients vector is:

$$\{C\} = \begin{Bmatrix} c_j \\ c_k \end{Bmatrix} \tag{5.19}$$

The determinant of the matrix [A] is given by:

$$|A| = \begin{vmatrix} a_{j1} & a_{j2} \\ a_{k1} & a_{k2} \end{vmatrix} = a_{j1}\, a_{k2} - a_{j2}\, a_{k1} \tag{5.20}$$

If |A| is not zero then there is a unique solution for the (5.17) simultaneous equations and the corresponding intersection point coordinates can be obtained, using the Cramer's rule, from the following expressions:

$$xi = \frac{\begin{vmatrix} c_j & a_{j2} \\ c_k & a_{k2} \end{vmatrix}}{|A|}, \qquad yi = \frac{\begin{vmatrix} a_{j1} & c_j \\ a_{k1} & c_k \end{vmatrix}}{|A|} \tag{5.21}$$

If the intersection point lies on the boundary lines j and k between the boundary corners (see figure 5.17) then the following logical conditions must be true:

$$x_{j1} \le xi \le x_{j2} \quad \text{and} \quad y_{j1} \le yi \le y_{j2}$$
$$x_{k1} \le xi \le x_{k2} \quad \text{and} \quad y_{k1} \le yi \le y_{k2} \qquad (5.22)$$

Before testing expressions (5.22) it is necessary to guarantee that the xx and yy values of the boundary corner coordinates are in ascending order. In the example of figure 5.17 it was necessary to swap the values of y_{k1} with y_{k2} before using them in expressions (5.22).

If the determinant |A| is zero then the two boundary lines j and k are parallel or coincident. They will be parallel if the determinants in (5.21) are different from zero and coincident in the other case. If they are coincident it is necessary to check if the boundary line segments overlap. The conditions (5.22) can be used to test each boundary corner in turn.

Restart definition

When this option is selected the current warehouse boundary is erased and a new boundary can be created.

. Zoom window

This option changes the size of the display window. It is useful for improving the drawing level detail by increasing the physical coordinates resolution. The window concept is explained in 5.3.2.1. and a visual interpretation is given by the scheme of figure 5.11. The changes in the coordinate transformations are handled by the equations (5.7) and (5.8) by updating the window extreme coordinates. Another important concept involved with the windowing transformations is the clipping concept. When defining a window that frames only a part of the virtual address space (see figure 5.11) it is necessary to cut off all the graphic information outside the window boundaries. This process is called clipping. The Tektronix terminals local command set do the clipping automatically when the window coordinates are redefined.

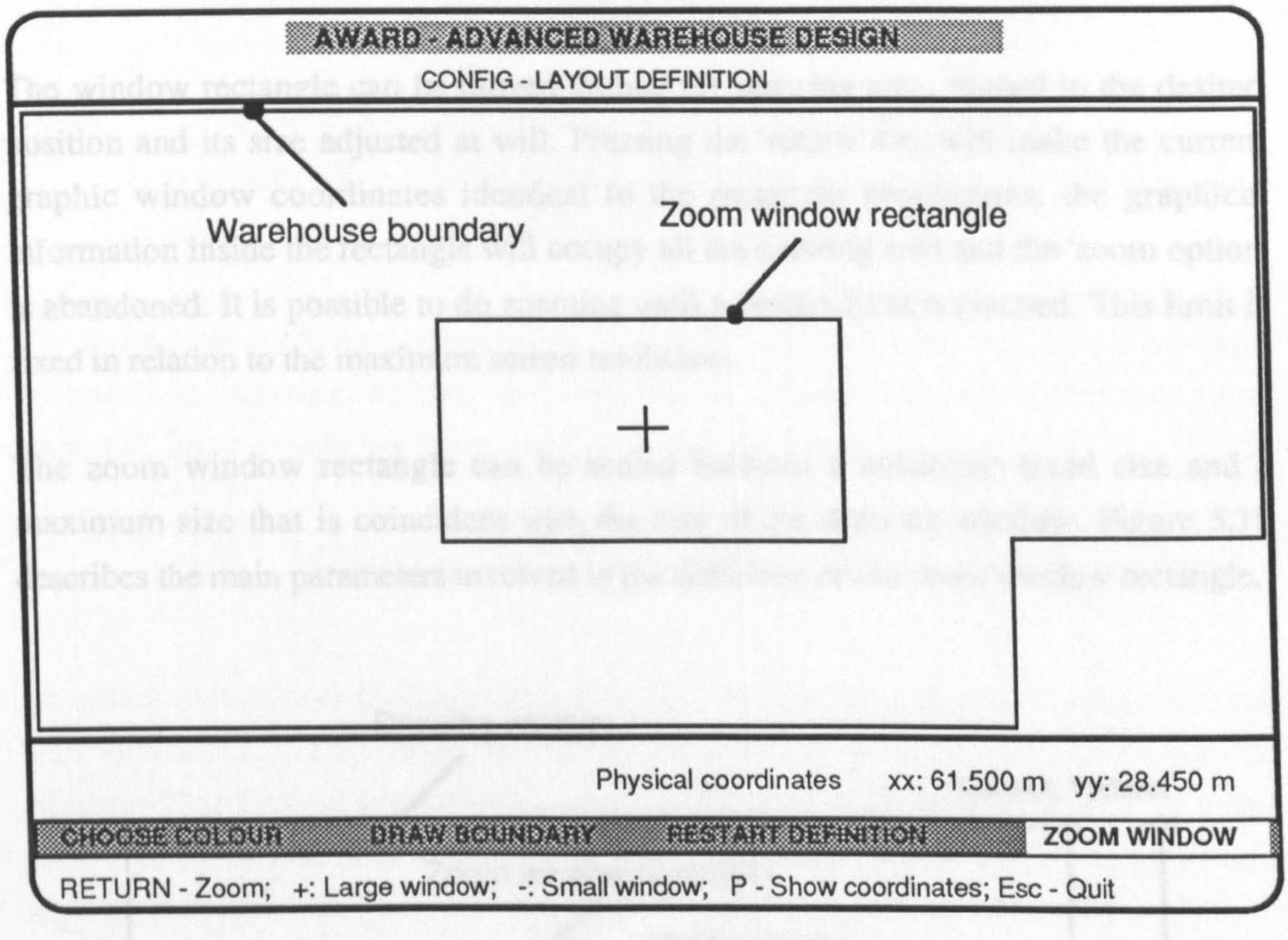


Figure 5.18 The zoom window option

After selecting the 'zoom window' option a rectangle is placed in the centre of the drawing area (see figure 5.18) and the following sub-menu is displayed at the bottom of the command area:

RETURN - Zoom: change the display window to the size of the window rectangle and quits the option;

+: Large window: enlarge the window rectangle;

-: Small window: diminish the window rectangle;

P - Show coordinates: show the warehouse coordinates (metres) corresponding to the current graphic cursor position;

Esc - Quit: escape from the 'zoom window' option.

The window rectangle can be moved around the drawing area, placed in the desired position and its size adjusted at will. Pressing the 'return' key will make the current graphic window coordinates identical to the rectangle coordinates, the graphical information inside the rectangle will occupy all the drawing area and the 'zoom option' is abandoned. It is possible to do zooming until a certain limit is reached. This limit is fixed in relation to the maximum screen resolution.

The zoom window rectangle can be scaled between a minimum fixed size and a maximum size that is coincident with the size of the drawing window. Figure 5.19 describes the main parameters involved in the definition of the zoom window rectangle.

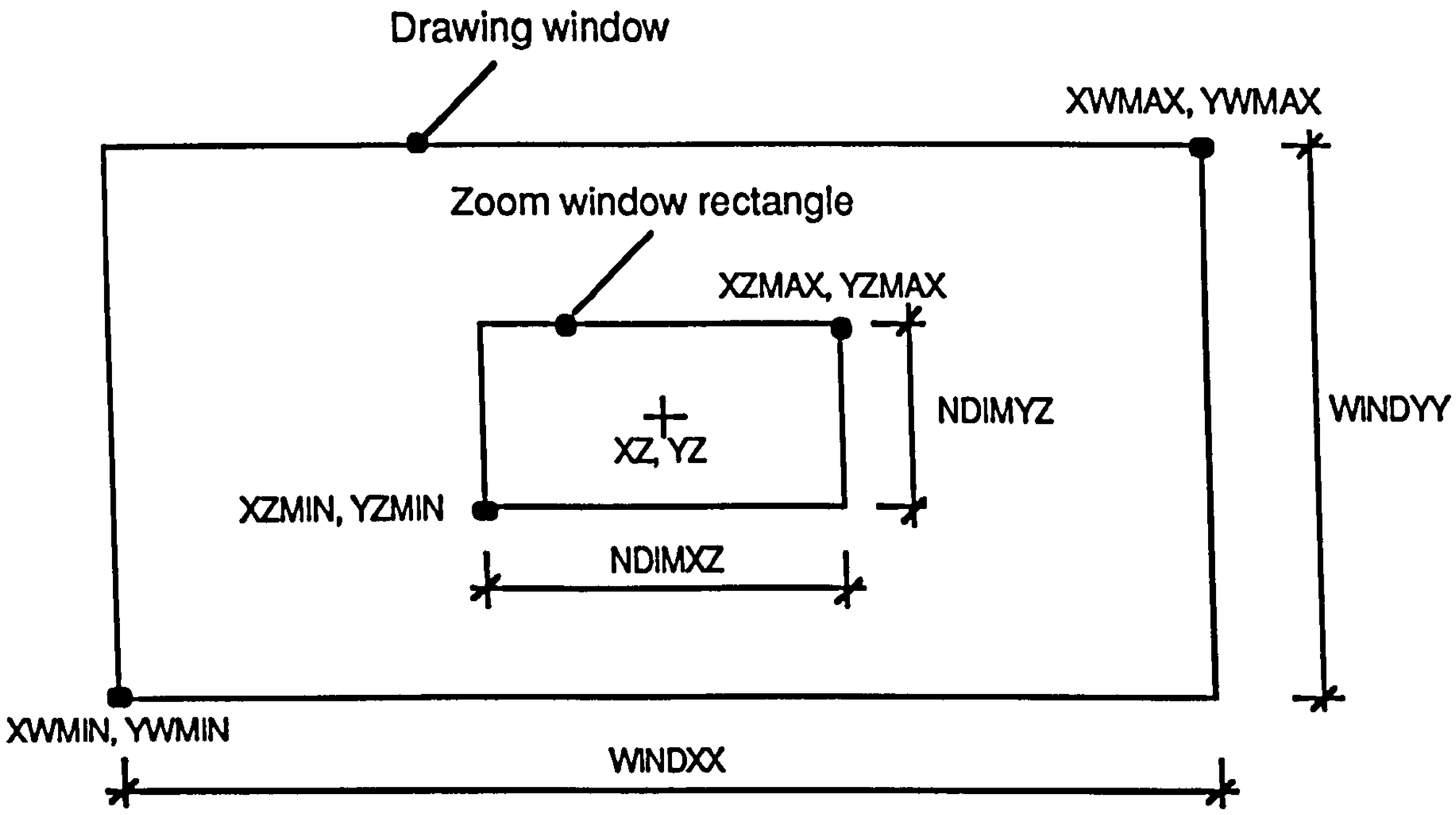


Figure 5.19 Main parameters in the definition of the zoom window rectangle

The scaling of the zoom window rectangle is made by incrementing or decrementing the value of a scale factor that is used to compute the new rectangle dimensions. The minimum and maximum values for the zoom scale factor are:

$$ZSCMIN = 1$$

$$ZSCMAX = \frac{WINDXX}{NDIZSE} \tag{5.23}$$

where WINDXX is the drawing window XX dimension;
NDIZSE is the minimum size for the XX zoom rectangle dimension in the Tektronix virtual coordinates.

The zoom scale increment is calculated to offer a range of approximately fifty positions between the drawing window size and the minimum size of the zoom window rectangle. So, the zoom increment value will be equal to:

$$ZSCINC = \frac{ZSCMAX - ZSCMIN}{50} \tag{5.24}$$

To assure that there is no distortion when zooming it is necessary to maintain the ratio between the XX and YY dimensions in the zoom window as the same as in the drawing window. This is done scaling only the zoom rectangle XX dimension and calculating the YY dimension in such a way that the same ratio is maintained. The drawing window ratio is equal to:

$$XYRATI = \frac{WINDYY}{WINDXX} \tag{5.25}$$

where WINDXX, WINDYY are the drawing window dimensions,

and the zoom window dimensions are obtained using the following expressions:

$$NDIMXZ = ZSCALE \times NDIZSE$$

$$NDIMYZ = XYRATI \times NDIMXZ \tag{5.26}$$

where NDIMXZ is the XX zoom window rectangle dimension;
NDIMYZ is the YY zoom window rectangle dimension;
ZSCALE is the current zoom scale (in the range given by (5.23): ZSCMIN ≤ ZSCALE ≤ ZSCMAX).

The default size for the zoom window rectangle is one third (see figure 5.18) of the drawing window size and its dimensions are obtained from expressions (5.26) by setting the zoom scale factor ZSCALE equal to one third of its maximum value. The default position for the zoom window rectangle is the centre of the drawing window. The centre coordinates (see figure 5.18) are obtained from the following expressions:

$$XZ = \frac{XWMIN + XWMAX}{2} \qquad (5.27)$$

$$YZ = \frac{YWMIN + YWMAX}{2}$$

where XWMIN, XWMAX are the drawing window XX extreme coordinates;
YWMIN, YWMAX are the drawing window YY extreme coordinates.

In figure 5.20 it is given an overall description of the algorithm to implement the zoom facility.

. Restore window

This option restores the drawing window to its initial size and updates the graphic screen.

. End definition

When this option is selected the warehouse boundary is, if necessary, automatically closed and after checking that the definition is valid the control returns to the main menu.

. Abort definition

After selecting this option and confirming the choice the configuration module is abandoned and all the work done since the beginning of the session or since the last save is discarded.

```
Display the zoom menu
WHILE last key pressed is not "RETURN" or "ESC"
    CALL a subroutine to enable moving the zoom rectangle and return when a key is pressed
    IF the key pressed is the "RETURN" key THEN
        Make the drawing window the same size as the zoom rectangle window
        Update the graphic screen
    ELSE IF the key pressed is "P" THEN
        Compute the world coordinates corresponding to the centre of the zoom rectangle
        Display the world coordinates (in metres)
    ELSE IF the key pressed is "+" THEN
        Increment the zoom scale
        Compute the new rectangle dimensions
        IF the zoom rectangle is inside the drawing window THEN
            Scale the zoom rectangle and update it on the screen
        ELSE
            Beep the terminal bell
        END IF
    ELSE IF the key pressed is "-" THEN
        IF the zoom scale is greater than the minimum value THEN
            Decrement the zoom scale
            Compute the new rectangle dimensions
            IF the zoom rectangle is inside the drawing window THEN
                Scale the zoom rectangle and update it on the screen
            ELSE
                Beep the terminal bell
            END IF
        END IF
    ELSE IF the key pressed is not "ESC" THEN
        Beep the terminal bell
    END IF
END WHILE
```

Figure 5.20 Algorithm for implementing the zoom facility

5.3.2.1.2. Reception and despatch areas

The warehouse reception and despatch areas are defined in a similar way to the warehouse boundary. Equivalent menus to the one presented in figure 5.15 are used to enable the reception and despatch areas definition. When the reception or the despatch boundaries are closed either manually or automatically by choosing "end definition" the area inside the boundary is painted with the selected colour. This is useful to highlight the different areas inside the warehouse. Apart from the visual aspects the reception and despatch areas are also used to find the potential points, in the transporters path network, that can be selected later for creating reception or despatch bays.

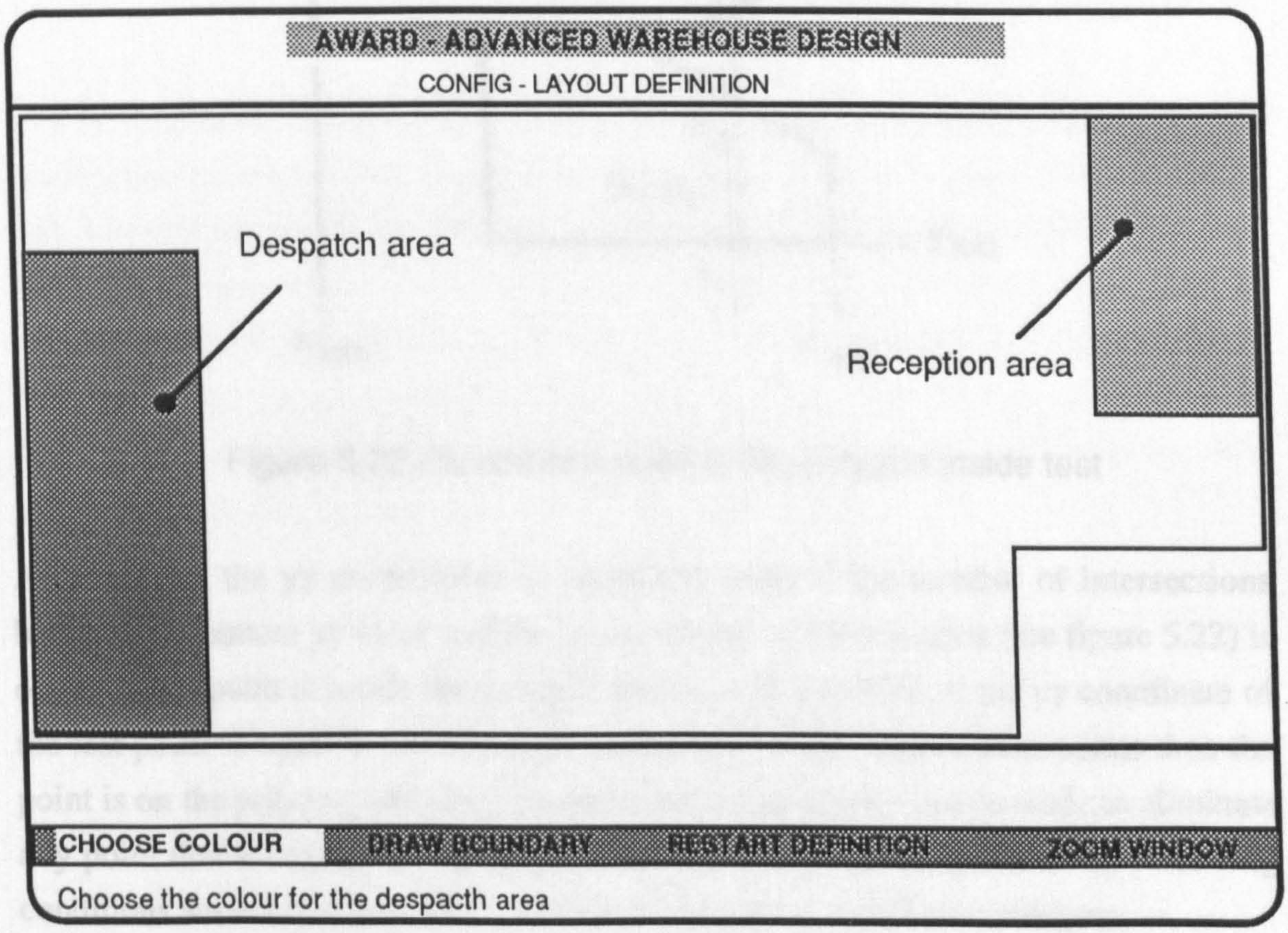


Figure 5.21 The warehouse layout showing the reception and despatch areas

Figure 5.21 shows the warehouse layout after the reception and despatch areas have been defined. In this case, as with the warehouse boundary, it is possible to define any polygonal line providing that none of its sides intersect each other or the warehouse boundary. All the checks are made on line and the terminal bell will sound everytime

that an attempt is made to set an incorrect entry. When setting the first corner, of the reception or despatch boundaries, an additional check is made to prevent it being set outside the warehouse boundary.

During the configuration module it is often necessary to test if a point is inside, outside or on the boundary of a polygon. One method to do that is to take the vertical line that contains the test point and find the intersections between this line and the polygon edges (see figure 5.22).

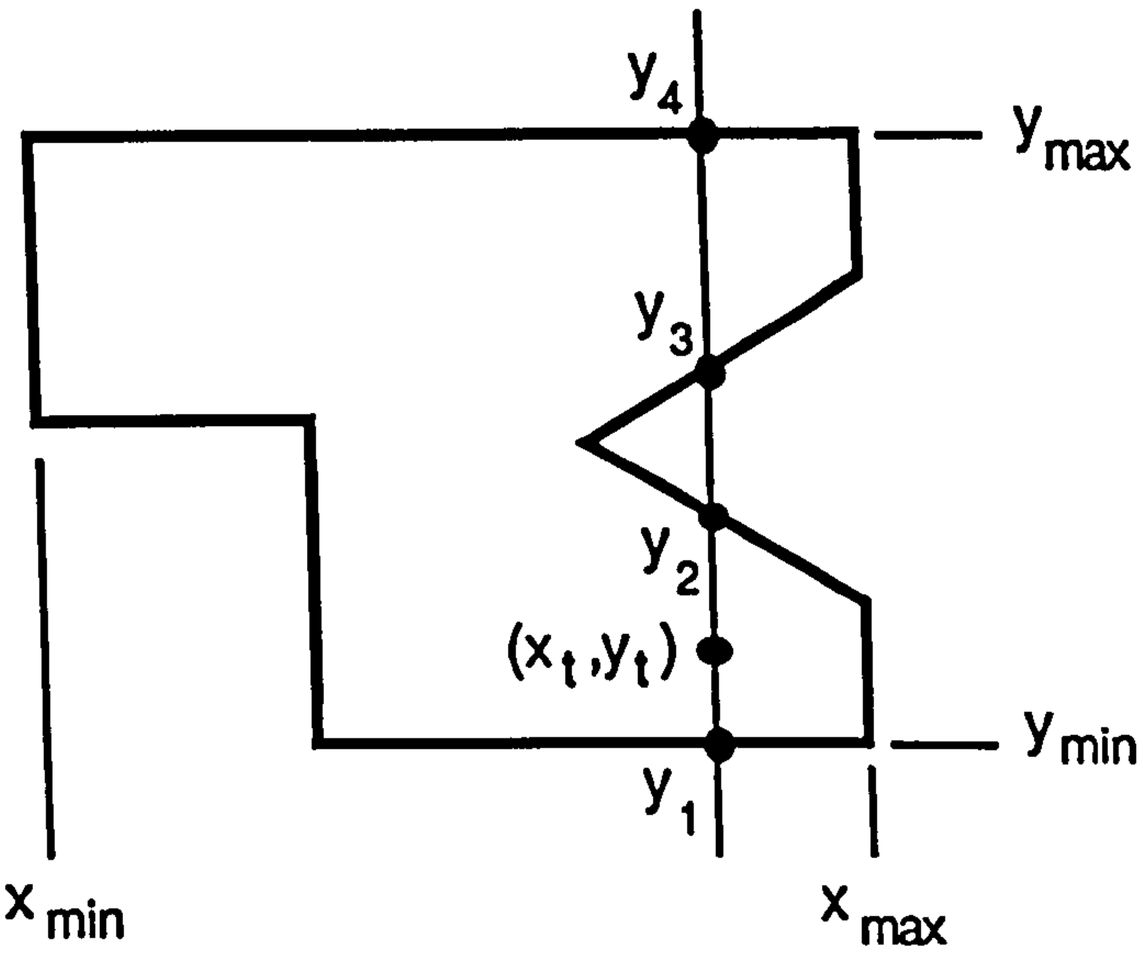


Figure 5.22 Parameters used in the polygon inside test

After sorting the yy coordinates in ascending order if the number of intersections between the bottom yy value and the yy coordinate of the test point (see figure 5.22) is odd then the point is inside the polygon otherwise it is outside. If the yy coordinate of the test point is equal to one of the yy coordinates of the intersections points then the point is on the polygon boundary. Furthermore, an initial test can be made to eliminate any point that is outside the rectangle containing the polygon. If any of the following conditions are not true (see figure 5.22) then the point is outside the polygon:

$$x_{min} \le x_t \le x_{max} \,, \quad y_{min} \le y_t \le y_{max} \tag{5.28}$$

When computing the intersection points of figure 5.22 some situations arise that can cause problems to the use of the method described. The first situation is when the vertical line, containing the test point, is coincident with one or more polygon edges (see figure 5.23).

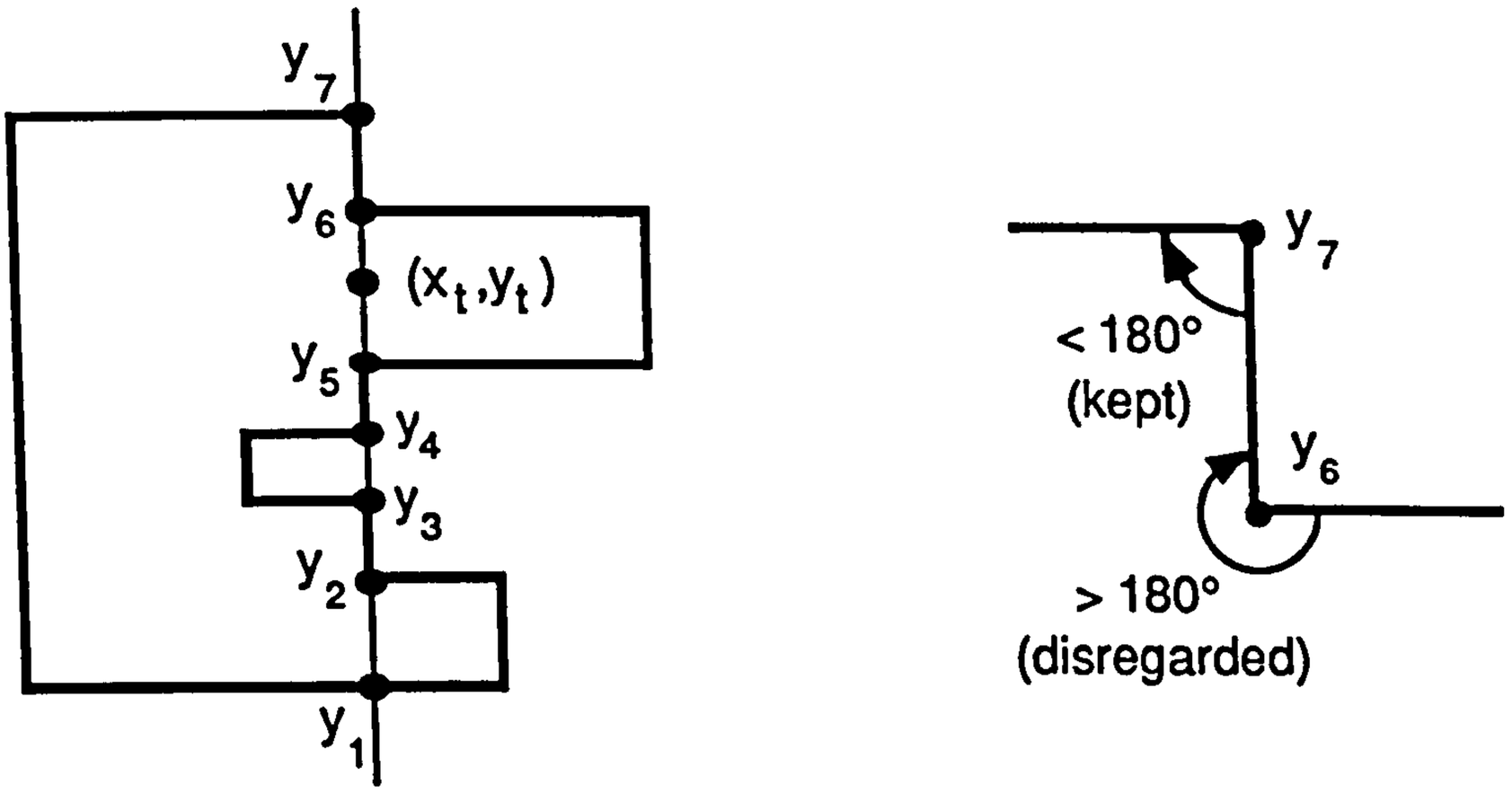


Figure 5.23 Rule for testing points when there is coincidence with polygon edges

In this situation it is necessary, for each coincident edge, to decide for each of the two intersection points whether they should be kept or not in the list used to do the inside test. The procedure is to compute the internal angles, corresponding to the two vertices of the coincident edge, and if the angle is greater then 180° (see figure 5.23) the intersection point is disregarded otherwise it is kept. In the example of figure 5.23 the intersection points y_2, y_5 and y_6 would be disregarded. So, the test point (x_t, y_t) is found to be located inside the polygon because the number of intersections points with yy coordinates less than y_t (y_1, y_3 and y_4) is an odd number.

For computing the angles between polygon edges and for other purposes during the configuration module, it is useful to know the direction cosines for each polygon edge. The xx and yy direction cosines can be obtained from the following expressions:

$$dcx = \frac{x_{i+1} - x_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \qquad (5.29)$$

$$dcy = \frac{y_{i+1} - y_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}$$

where x_i, y_i are the polygon edge initial vertice coordinates;

x_{i+1}, y_{i+1} are the polygon edge final vertice coordinates.

To find the angle between two consecutive polygon edges two vectors defined by the coordinates of three consecutive polygon vertices are used as described in figure 5.24.

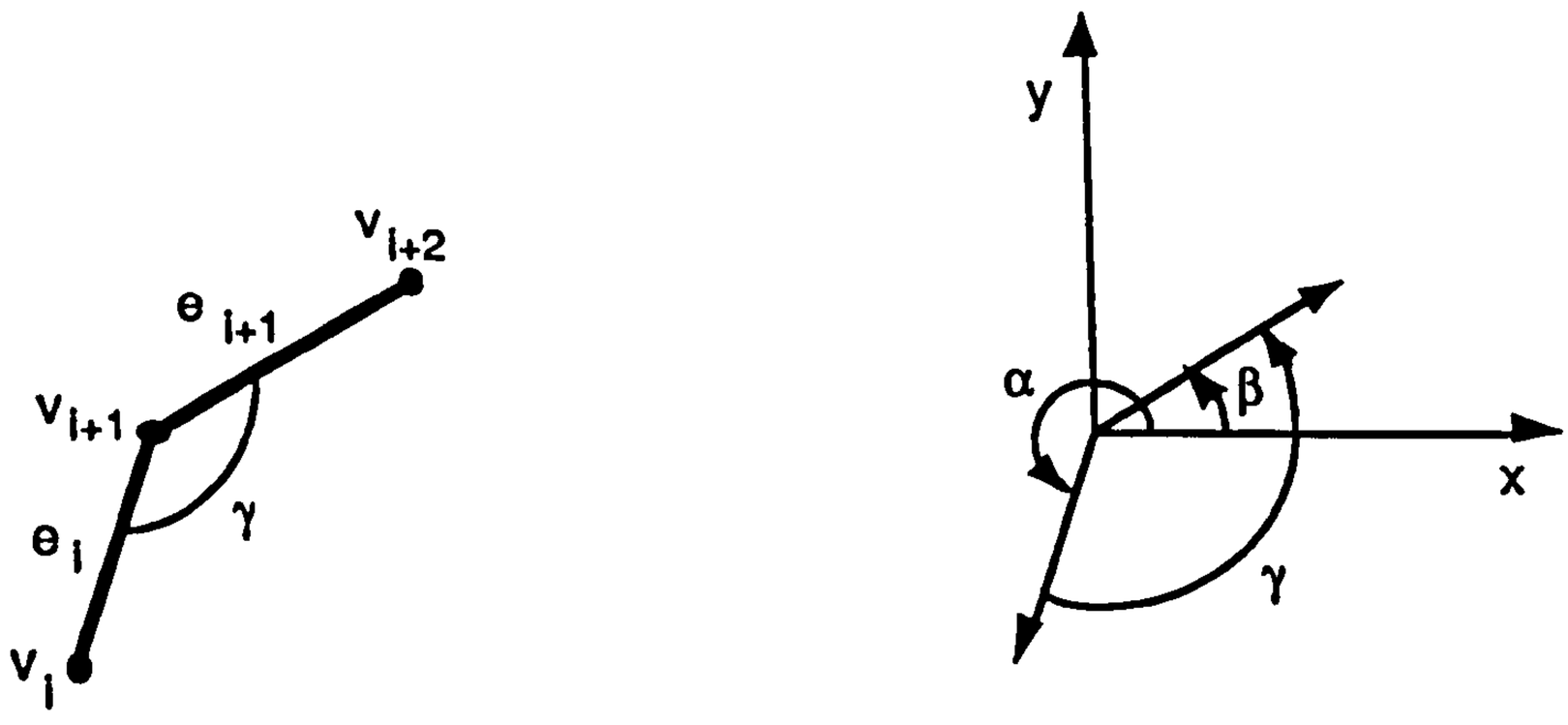


Figure 5.24 Parameters for computing the angle between consecutive polygon edges

The direction cosines (dcx_i, dcy_i) and (dcx_{i+1}, dcy_{i+1}) for edges e_i and e_{i+1} are given by expressions (5.29) using the coordinates of the corresponding vertices. The two angles α and β defined between the two vectors (see right scheme of figure 5.24) and the xx direction can be obtained from the following expressions:

$$\alpha' = \frac{180 \arccos(-dcx_i)}{\pi} \quad \left| \begin{array}{ll} \alpha = \alpha' & (-dcy_i) \geq 0 \\ \alpha = 360° - \alpha' & (-dcy_i) < 0 \end{array} \right.$$

$$\beta' = \frac{180 \arccos(dcx_{i+1})}{\pi} \quad \left| \begin{array}{ll} \beta = \beta' & dcy_{i+1} \geq 0 \\ \beta = 360° - \beta' & dcy_{i+1} < 0 \end{array} \right. \qquad (5.30)$$

this includes the transformation from radians to degrees. If one of the angles is from the third or fourth quadrant, what happens when $(-dcy_i) < 0$ for angle α and $dcy_{i+1} < 0$ for angle β, then it is necessary to take the complementary angle. This is done by subtracting from 360° the value given by expressions (5.30) for α' and β'. The angle between the consecutive polygon edges can then be easily calculated using the angles α and β:

$$\begin{array}{ll} \gamma = 360° - \alpha + \beta & \alpha > \beta \\ \gamma = \beta - \alpha & \alpha \leq \beta \end{array} \qquad (5.31)$$

After calculating the angles for all polygon vertices, an additional test is made to check if they correspond to the polygon internal or external angles. This is done by comparing the sum of the calculated angles with the sum of their complements. The angles taken in the end are those whose sum is smaller.

Returning to the polygon inside test, another particular situation that can arise is when the vertical line that contains the test point crosses one or more polygon vertices without being coincident with the polygon edges. In this situation it is necessary again to determine if the intersection point should be kept or discarded. This will depend on the relative position of the adjacent polygon edges (see figure 5.25).

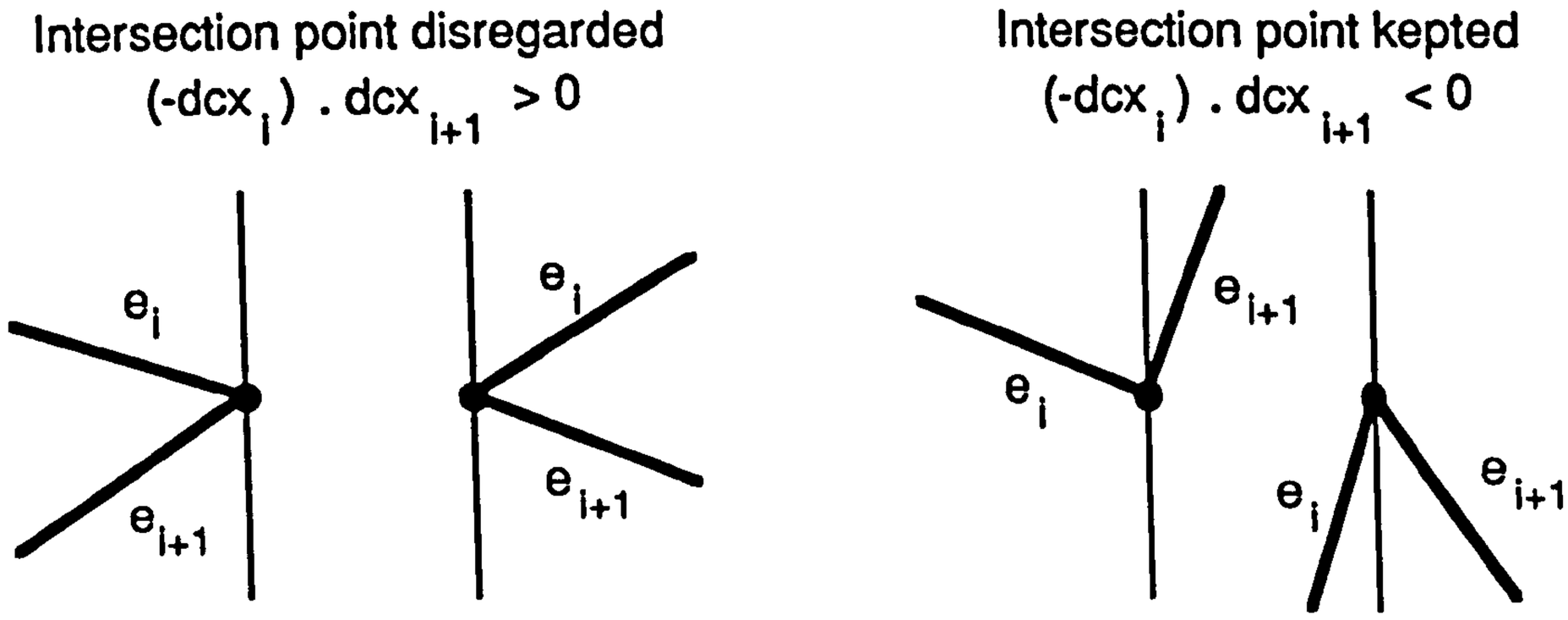


Figure 5.25 The different situations when a vertice is crossed in the inside test

If the adjacent polygon edges are on the same side of the vertical line (see left of figure 5.25) then the vertical line before and after crossing the vertice stays in the same position in relation to the polygon. This means that the intersection point should be disregarded. On the other hand, if the polygon edges are in opposite sides of the vertical (see right of figure 5.25) then there is a change in the position of the vertical line in relation to the polygon. This means that the intersection point should be kept. One way of knowing if the polygon edges are both on the same side of the vertical is to use the sign of the direction cosines from the vectors defined on the right of figure 5.24. If the sign of the xx direction cosine in both vectors is: negative, that means they are on the second or third quadrant so on the left of the vertical line; positive, that means they are on the first or fourth quadrant so on the right of the vertical line. Figure 5.26 gives an overall description of the algorithm that implements the inside polygon test.

```
Find the equation of the vertical line that contains the testing point
edge = first
Done = FALSE
WHILE not Done and edge less or equal last edge
    IF current edge is vertical THEN
        IF the vertical line is coincident with current edge THEN
            IF the testing point is on the polygon edge THEN
                Set the 'on polygon boundary flag'
                Done = TRUE
            ELSE
                Increment the number of coincident edges
                Record the current edge number
            END IF
        END IF
    ELSE
        Find the intersection point between the vertical line and the current edge
        IF the testing point is on the polygon edge THEN
            Set the 'on polygon boundary flag'
            Done = TRUE
        ELSE
            Increment the number of intersection points
            Add the yy coordinate of the intersection point to the list
            Add the current edge number to the list
        END IF
    END IF
    edge = edge + 1
END WHILE
IF not Done
    FOR each coincident edge
        FOR each vertice edge
            IF the vertice angle is greater than 180° THEN
                Take the vertice yy coordinate from the intersection point list
                Take the corresponding edge number from the list
                Decrement the number of intersection points
            END IF
```

```
        NEXT
    NEXT
    Sort the intersection points list and edge numbers list by ascending order of yy coordinates
    FOR each intersection point until the one before the last
        IF the current intersection point and the next one are coincident THEN
            IF the two corresponding edges are on opposite sides of the vertical THEN
                Take the current point from the intersection point list
                Take the corresponding edge number from the list
                Decrement the number of intersection points
            ELSE
                Take the current point and the next one from the intersection point list
                Take the corresponding edge numbers from the list
                Decrement by two the number of intersection points
            END IF
        END IF
    NEXT
    Counter = 0
    FOR each intersection point and not Done
        IF the testing point yy coord. is greater than the intersection point yy coord. THEN
            Counter = Counter + 1
        ELSE
            Done = TRUE
        END IF
    NEXT
    IF Counter is odd THEN
        Set the 'inside polygon flag'
    END IF
END IF
```

Figure 5.26 Algorithm for implementing the inside polygon test

5.3.2.1.3. No-go areas

The no-go areas are used to prevent anything being created in reserved areas such as offices, pillars etc. The no-go areas are defined as rectangles that can be placed

anywhere inside the warehouse. After creating a no-go area nothing can be defined inside, including another no-go area. Figure 5.27 shows the menu options available during the no-go areas definition.

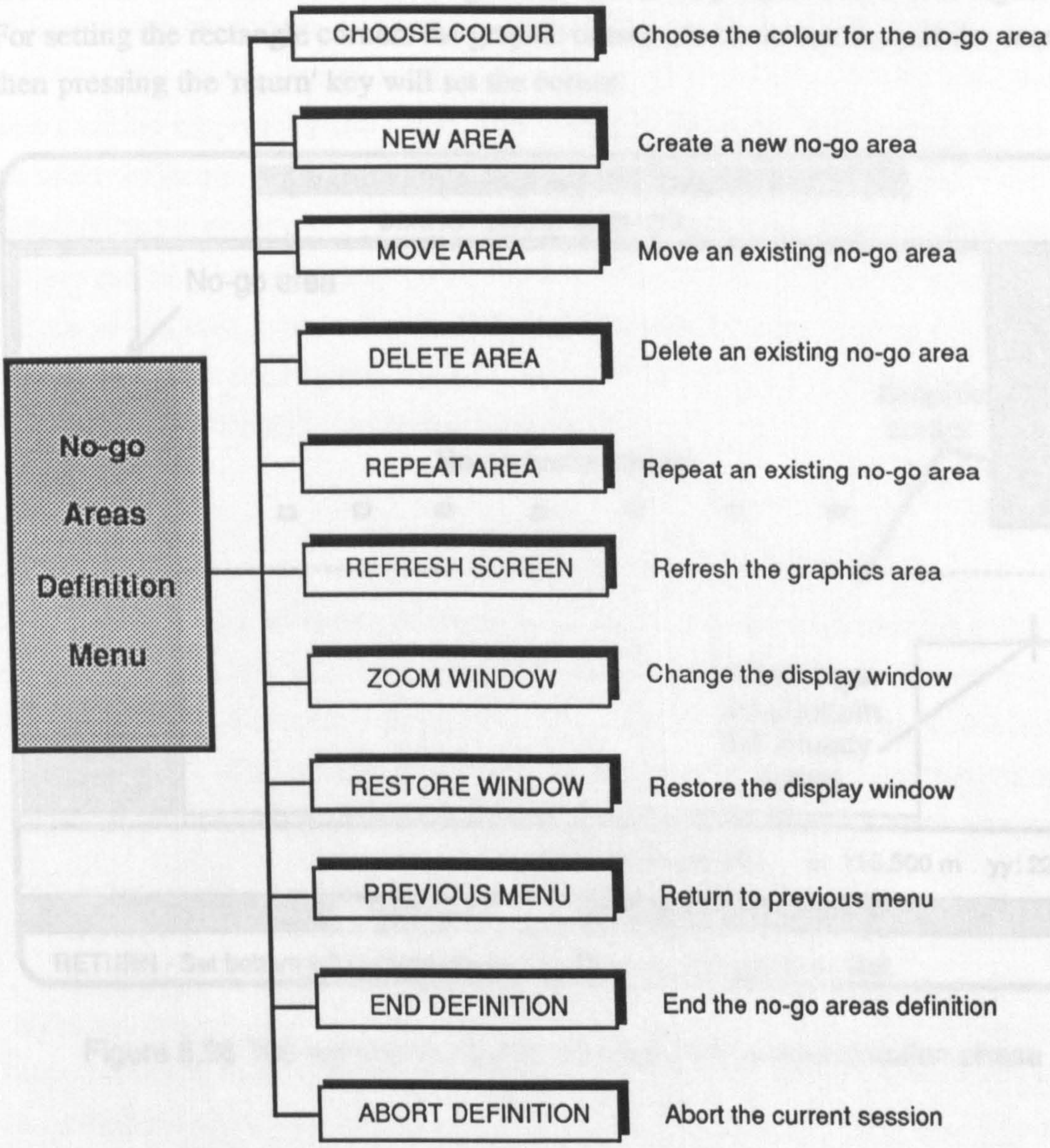


Figure 5.27 No-go areas definition menu

Some of these options were already described in 5.3.2.1.1. and because they work the same way through the different stages, they will not be described again.

. New area

With this option it is possible to create a new no-go area. The colour used is the current colour selected by the 'choose colour' option. A no-go area is defined by first setting the bottom left corner of the rectangle and then the top right corner (see figure 5.28). For setting the rectangle corners the graphic cursor can be moved around the screen and then pressing the 'return' key will set the corner.

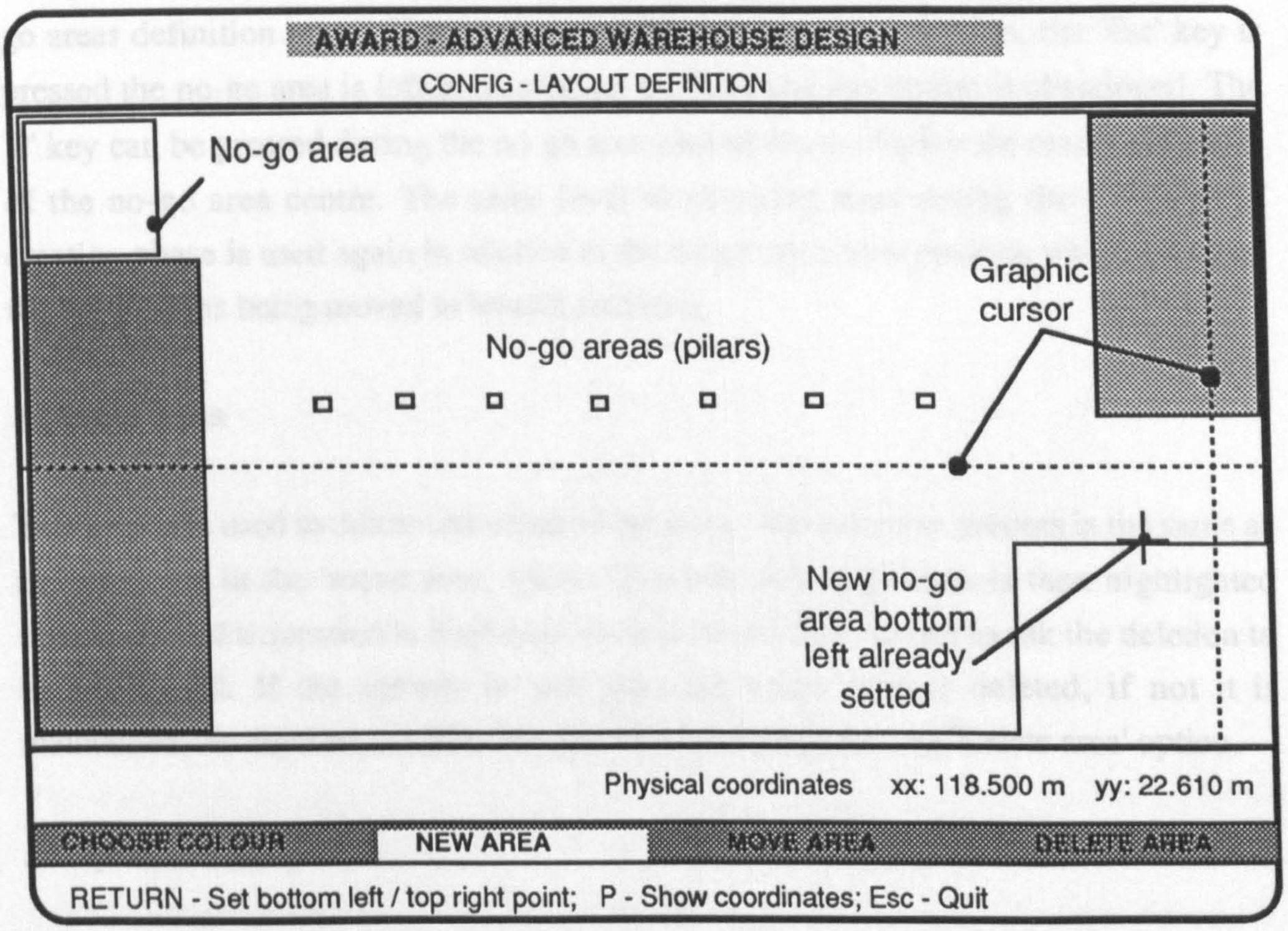


Figure 5.28 The warehouse layout showing a no-go area creation phase

At any time, by pressing the 'P' key, the real coordinates (metres) corresponding to the current graphic cursor position can be displayed (see figure 5.28). A no-go area can not be created outside the warehouse, neither inside or intersecting an existing block (reception area, despatch area, no-go areas, racking, transporter aisles etc.). An on-line check is made and the terminal bell will sound for each incorrect entry. After setting the no-go area top right corner the control returns to the no-go areas definition menu. The 'Esc' key is used to quit the 'New area' option without creating a no-go area.

. Move area

After creating a no-go area this option enables it to be moved to another place inside the warehouse. The first step is the selection of the no-go area to be moved. This is done using the graphic cursor and pressing the 'return' key. The selection algorithm consists of finding the no-go area which has the centre the nearest to the current graphic cursor position. The selected no-go area can then be moved around the screen and placed in a new position by pressing the 'return' key. This will also take the control back to the no-go areas definition menu. If at anytime, within the 'move area' option, the 'Esc' key is pressed the no-go area is left in its original position and this option is abandoned. The 'P' key can be pressed during the no-go area movement to display the real coordinates of the no-go area centre. The same level of checking used during the no-go areas creation phase is used again in relation to the no-go areas new position which prevents the no-go areas being moved to invalid positions.

. Delete area

This option is used to delete unwanted no-go areas. The selection process is the same as the one used in the 'move area' option. The selected no-go area is then highlighted (blinking) and a question is displayed at the bottom of the screen to ask the deletion to be confirmed. If the answer is 'yes' then the no-go area is deleted, if not it is maintained. At anytime the 'Esc' key can be used to abandon the 'Delete area' option.

. Repeat area

By using this option an existing no-go area can be duplicated any number of times. The no-go area selection process is the same as the one used in the 'move area' and 'delete area' options. After the selection, a copy of the no-go area can be moved around the screen, as in the 'move area' option, and by pressing 'return' a new copy will be placed at the current position. The 'Esc' key is used here to abandon the 'repeat area' option. Again the same level of checking is used, as in the previous options, to prevent setting no-go areas copies at invalid positions.

. Refresh screen

Sometimes during the drawing definition there is the necessity to update the screen. When this happens it is only necessary to select the 'refresh screen' option and the graphic screen will be erased and all graphic primitives redrawn.

. Previous menu

Selecting this option will take the control to the previous definition menu. In the case of the no-go area definition menu the control will return to the despatch area definition menu.

5.3.2.1.4. Racking modules

The racking modules are used as building blocks for the racking system. After defining the racking modules it is possible to select any of them to create a new racking. The racking will have the characteristics defined for that racking module type. For each racking module it is necessary to define its dimensions and the associated storage and access logic. In figure 5.29 is presented a schematic drawing of a racking showing the racking module and the dimensions that are needed to define it.

The racking module works as a single unit that when repeated, in the different directions, will enable the creation of the racking. So, when computing the racking module dimensions there must be taken into account the pallet clearances and the structure width. To each racking module is also associated a racking type. The racking type defines the way in which pallets are put away, picked and stored in the racking. The racking types included are:

. APR/SD - adjustable pallet racking / single deep;

. APR/DD - adjustable pallet racking / double deep;

. BLOCK STACKING;

. DRIVE-IN;

. **PMR** - powered mobile racking;

. **LIVE STORAGE.**

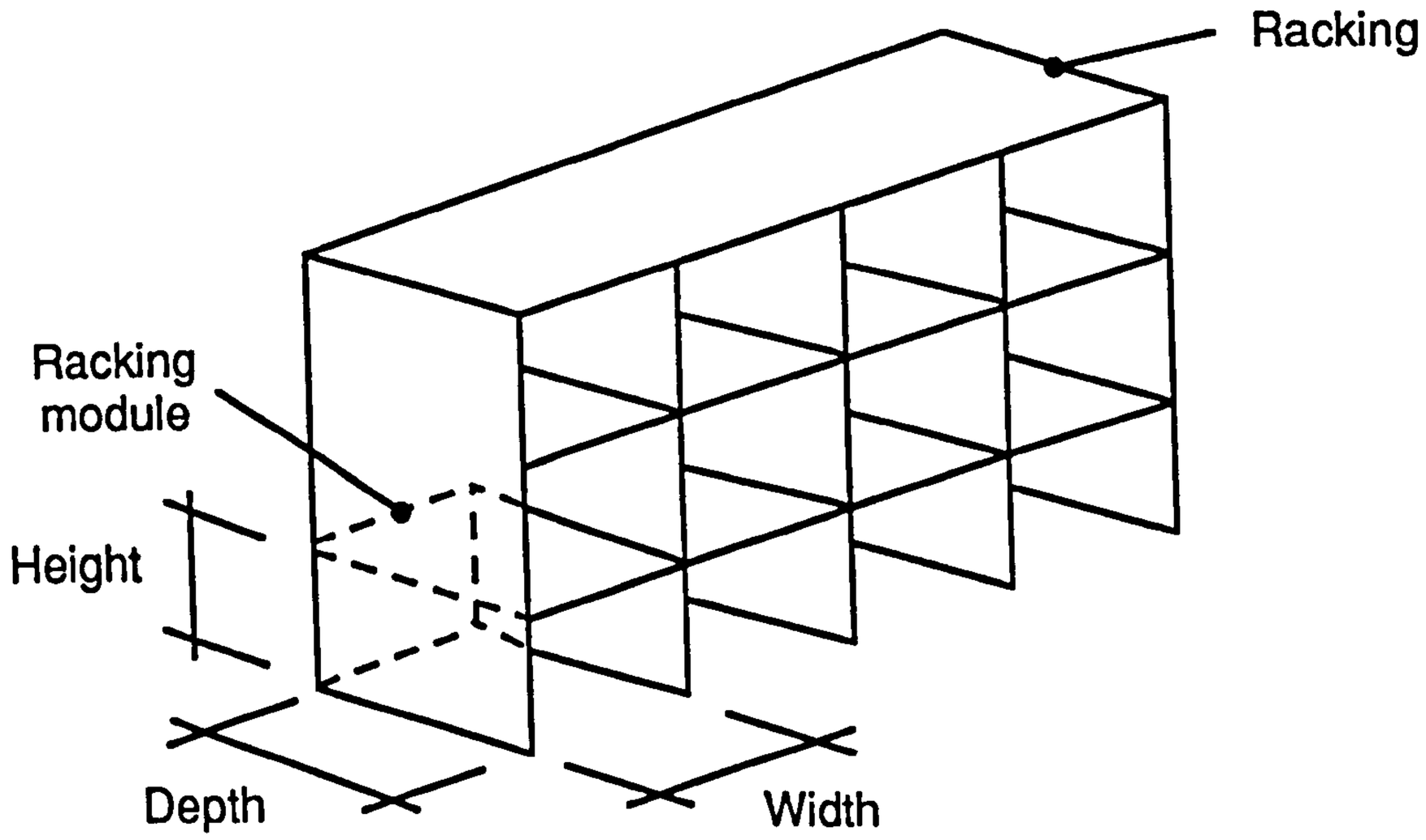
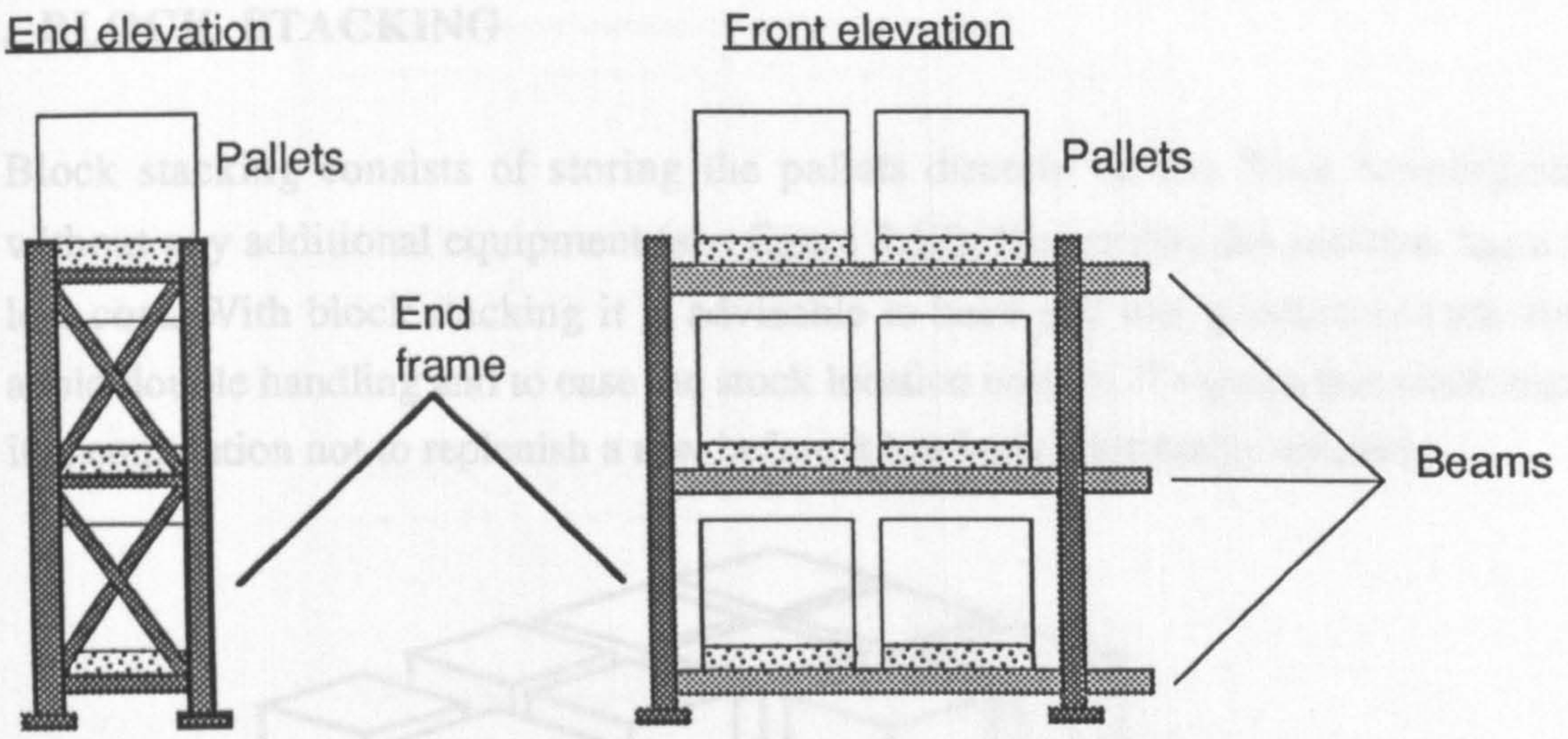


Figure 5.29 A warehouse racking showing the the racking module dimensions

These racking types cover most of the existing racking in palletised storage. Nevertheless any other type can be considered as being one of those types. That will be the case for small parts storage or other types of storage for specific types of load. The following gives a brief description of each of the racking types considered.

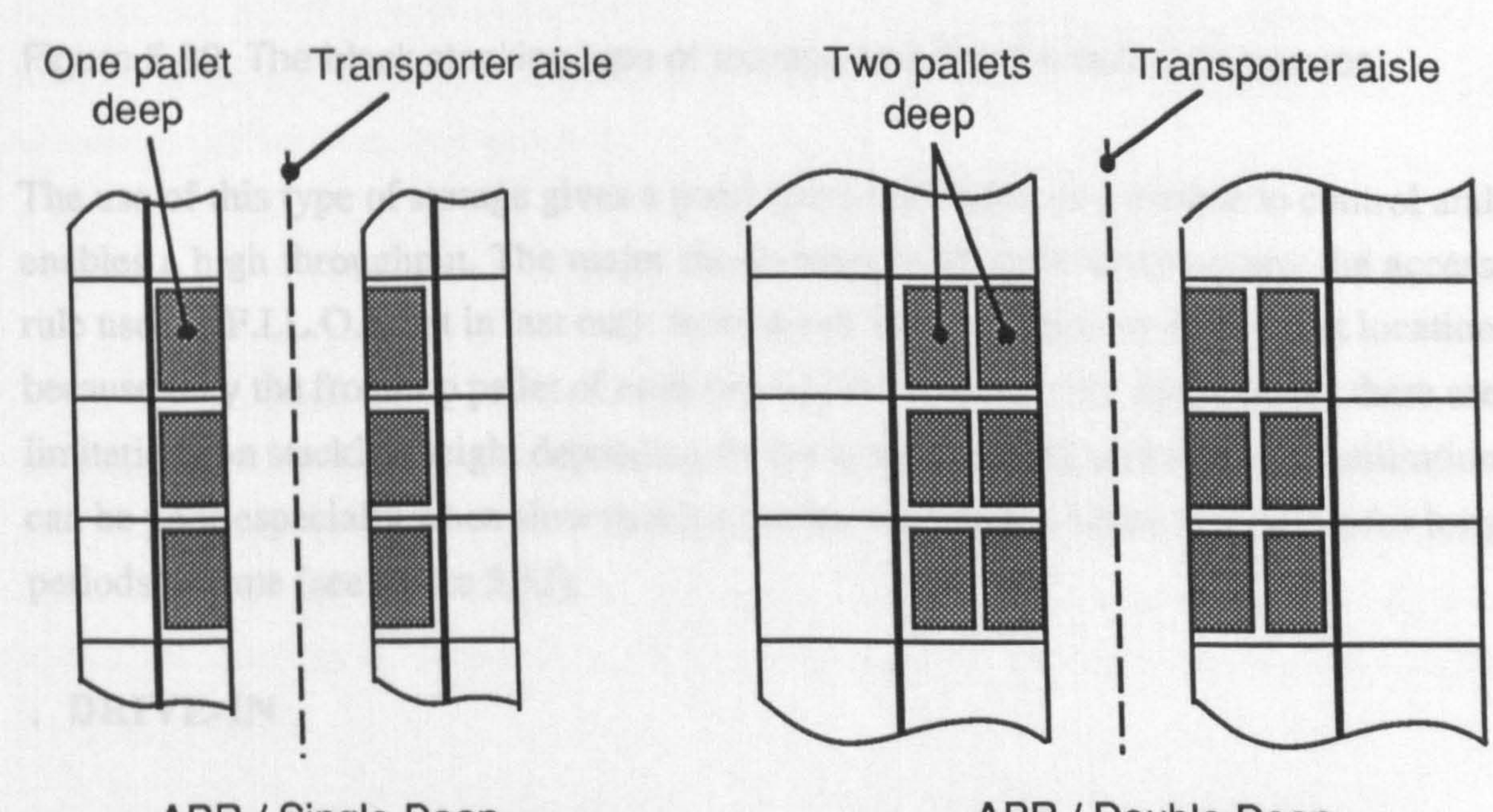
. **APR/SD and APR/DD**

Adjustable pallet racking is probably the most commonly used racking type. The racking is made up from horizontal beams and end frames (see figure 5.30). The pallet loads are supported by the beams which are supported by the uprights. The end frames are formed by two uprights with base plates, joined by horizontal and diagonal bracing members. The adjustable pallet racking/single deep permits a random access to every pallet location, it is not expensive and can handle different load heights. Perhaps its major drawback is a low overall utilisation of building volume due to the fact that one aisle only gives access to two rows of racking. One way to improve the utilisation level is to allow two pallets deep in each location using adjustable pallet racking/double deep.



5.30 Adjustable pallet racking single deep

With this type of racking the random access to all pallet locations is lost, special double deep trucks must be used and the F.I.F.O. (first in first out) rule is not full applicable. The use of this type usually assumes that in each location both pallets are of the same line item otherwise double handling is necessary to get access to the rear pallet.



5.31 Plan view of adjustable pallet racking single and double deep

. BLOCK STACKING

Block stacking consists of storing the pallets directly on the floor forming stacks without any additional equipment (see figure 5.32). Due to this the solution has a very low cost. With block stacking it is advisable to have just one product in each row to avoid double handling and to ease the stock location control. To guarantee stock rotation it is convention not to replenish a row before it has been completely cleared.

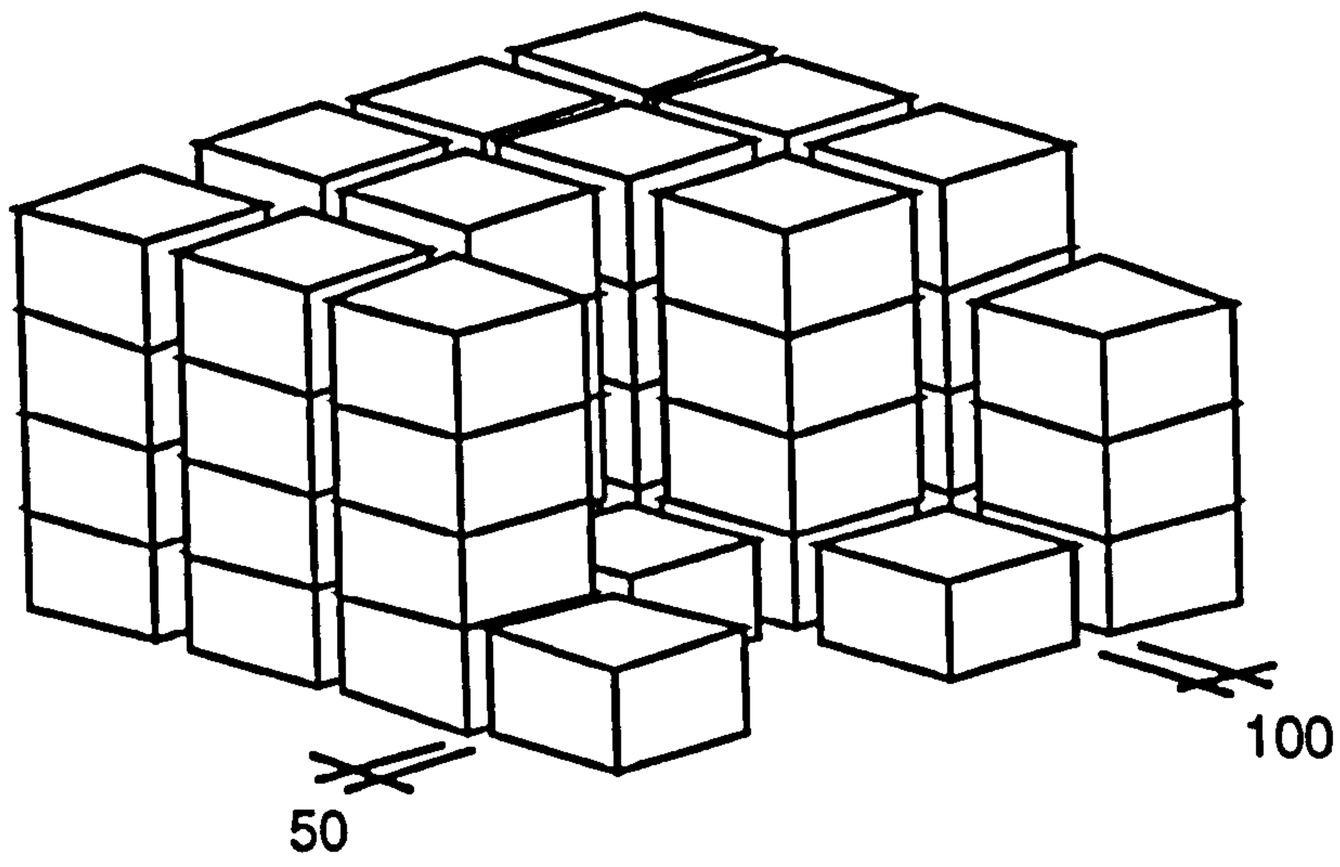
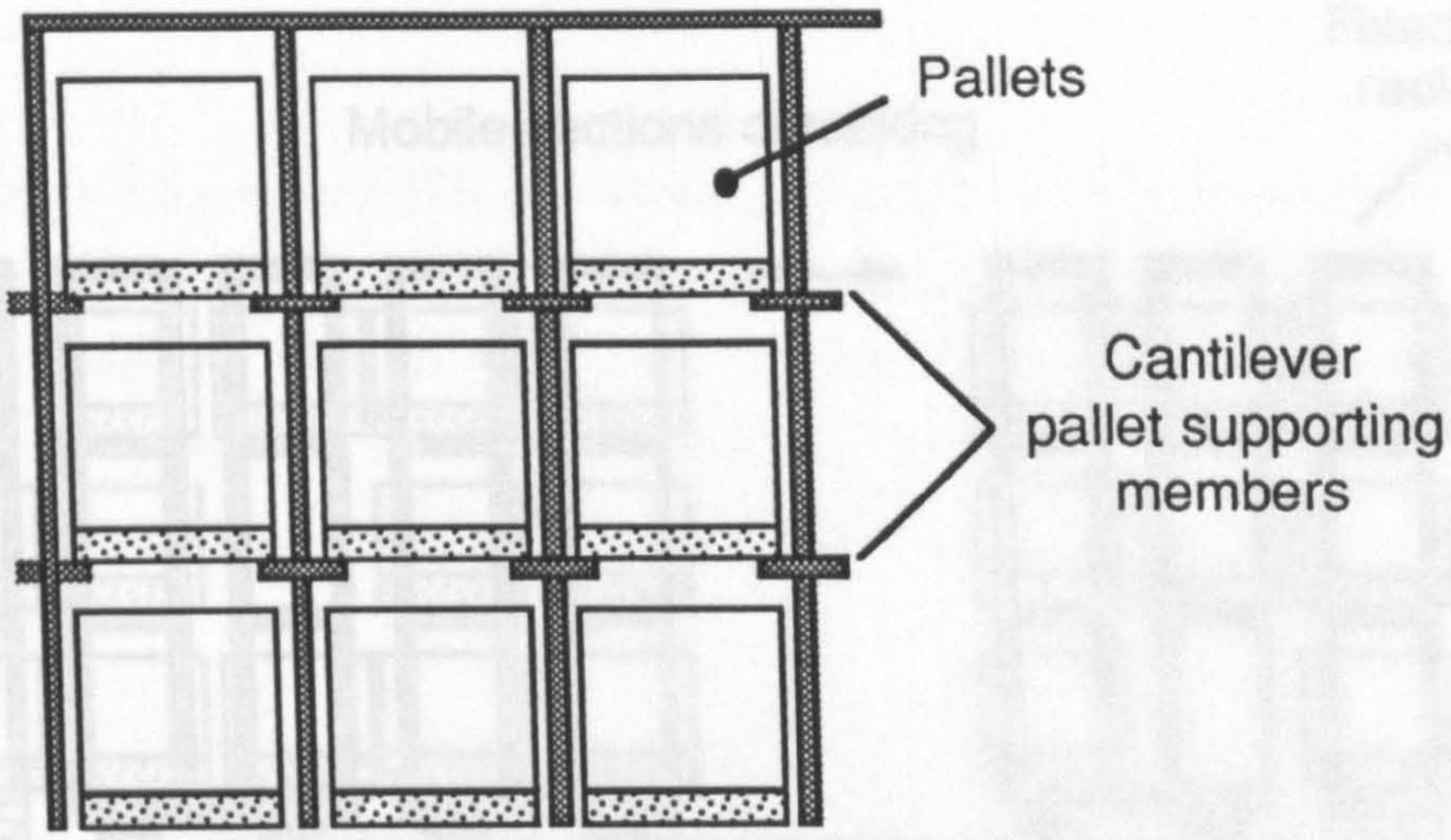


Figure 5.32 The block stacking type of storage showing the pallet clearances

The use of this type of storage gives a good space utilisation, it is simple to control and enables a high throughput. The major disadvantages of block stacking are: the access rule used is F.I.L.O. (first in last out); there is not random access to each pallet location because only the front/top pallet of each row can be accessed (see figure 5.30); there are limitations on stacking height depending on the product nature; and the space utilisation can be poor especially when slow moving products can cause holes in stacking for long periods of time (see figure 5.32).

. DRIVE-IN

The drive-in storage is operationally similar to block stacking. The difference between the two systems is that in block stacking the pallets support the weight of the pallets above and in drive-in there is a structure to support them (see figure 5.33).



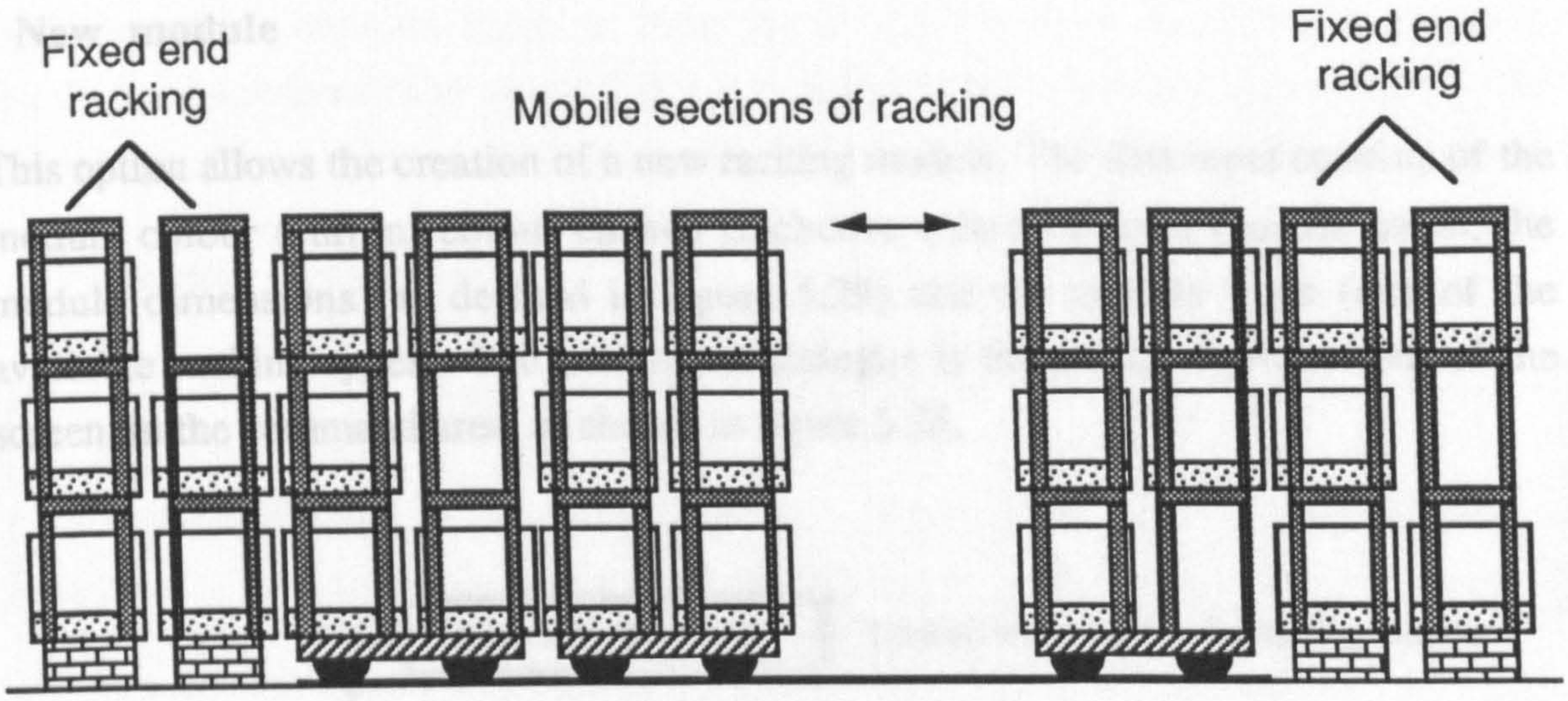
5.33 Drive-in racking front elevation

This structure consists of vertical frames joined together, in each floor, by cantilever pallet supporting members (see figure 5.33). This allows the truck to drive into the racking, between the vertical frames, and get access to the top/front pallet as in block stacking. The drive-in storage does not have the height restriction of block stacking because the pallet weight is supported by the structure. On the other hand the tight tolerances on the structure construction and on the flatness of the floor make this solution more complex and expensive to implement.

. PMR - Powered Mobile Racking

PMR storage gives very good space utilisation. This is achieved by using sections of adjustable pallet racking, close together, fixed on top of powered based frames which can move along rails (see figure 5.34). Because the racking can be moved to open an aisle to enable the truck to get in, only space for one aisle is required. Usually the racking is made up of a number of sections that can be moved between fixed end racks (see figure 5.34).

The powered mobile racking gives random access to every pallet location, as in the APR/SD storage, but with an increased access time due to the movement of the racking. This solution is expensive to implement but can be balanced with the better space utilisation which can lead to lower building size.



5.34 Powered mobile racking

. LIVE STORAGE

The live storage racking is made up of several levels of adjacent rows of inclined lanes with gravity roller conveyors. The racking input and output are on opposite sides. When a pallet is put away it will move along the roller conveyer until it reaches the end, if the row is empty, or the last pallet in the row. Because of the way it operates, the live storage access rule is always F.I.F.O. (first in first out). Only the pallets in the front of each row, at the lower end of the racking, can be accessed. This forces all pallets in one row to be of the same product. This system is well suited to high throughput. The installation is expensive and the space utilisation is not always good.

Any of the different types of racking, described above, can be used for defining the racking module logic during its creation. Figure 5.35 shows the menu options available during that stage.

Again, as most of the options were already described only the new ones concerning the racking modules definition will be described.

. New module

This option allows the creation of a new racking module. The data input consists of the module colour (current colour chosen in choose colour option), module name, the module dimensions (as defined in figure 5.29) and the module logic (one of the available racking types). The data entry dialogue is displayed at the bottom of the screen, in the command area, as shown in figure 5.36.

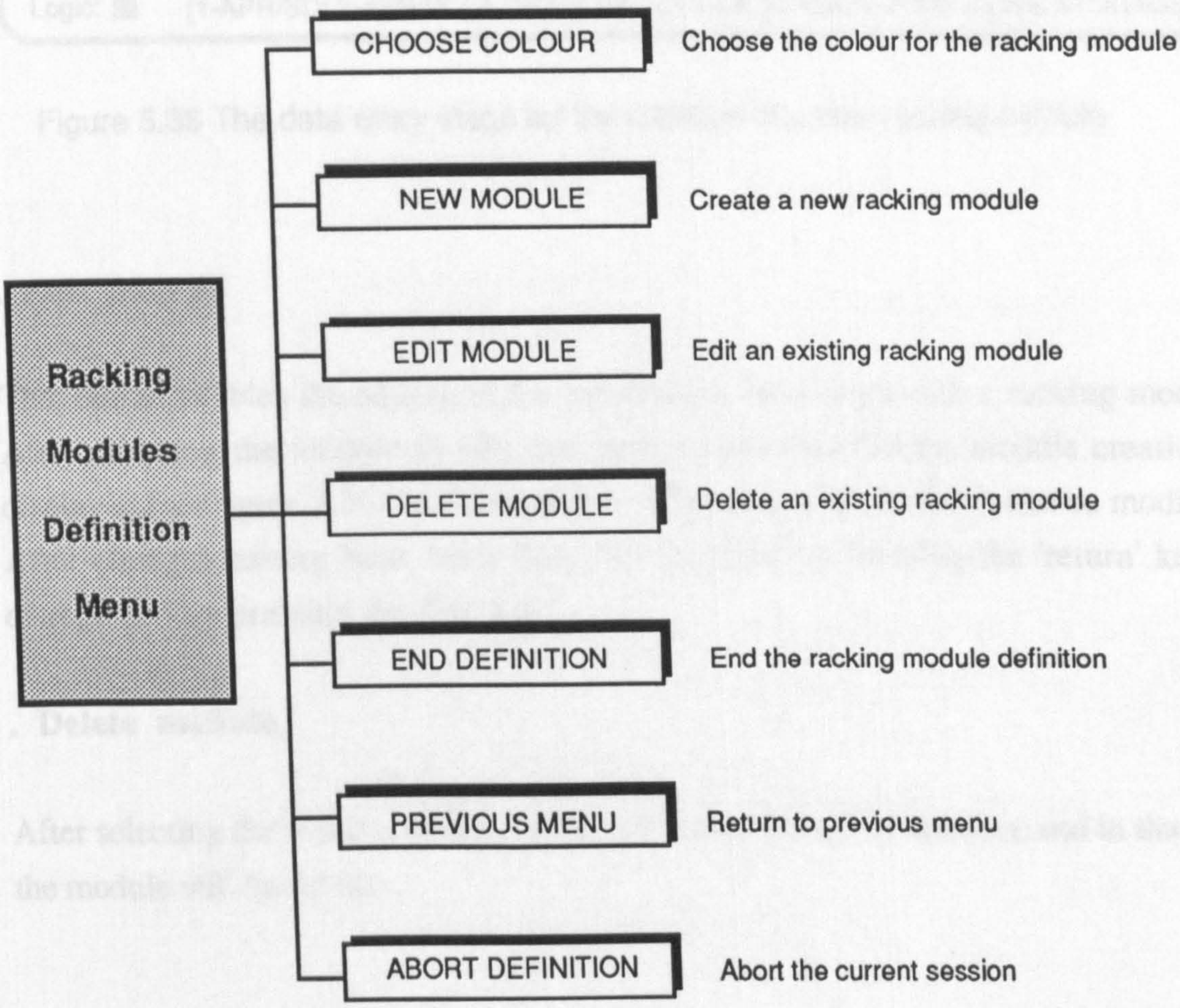


Figure 5.35 Racking modules definition menu

If the PMR racking type is chosen then an additional value needs to be entered after the first screen. This value, is called the mean access time to PMR (s), and specifies the truck delay before it can get into the racking. This delay is due to the movement of the racking to open an aisle to enable the truck to have access to a particular pallet (see figure 5.34). After the creation of a new module its scaled representation is displayed at the top of the command area (see figure 5.36) together with the module name. This is

used, during the creation stage, to show the existing modules and on later stages to enable the selection of the module to use as default. This selection is made using the arrow keys to highlight the desired module and then pressing the 'return' key.

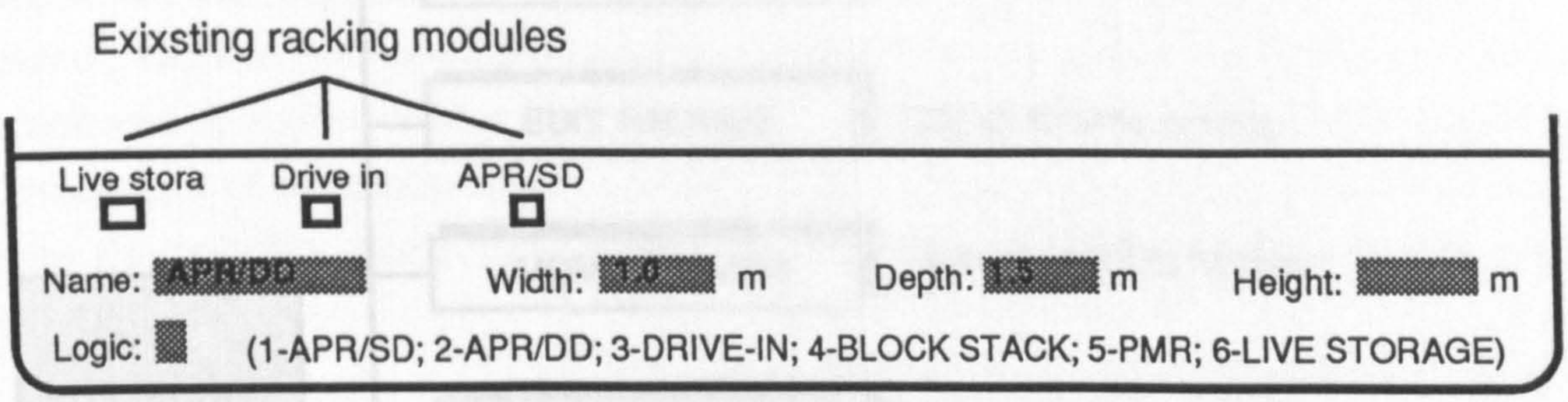


Figure 5.36 The data-entry stage for the creation of a new racking module

. Edit module

This option enables the editing of the information associated with a racking module. After selecting the module to edit, the same screen used for the module creation is displayed (see figure 5.36) with the module data and any of the fields can be modified. After changes having been made they can be saved by pressing the 'return' key or disregarded by pressing the 'Esc' key.

. Delete module

After selecting the racking module one is asked to confirm the deletion, and in that case the module will be deleted.

5.3.2.1.5. Racking definition

After having defined the racking modules they can be used to build the racking layout. Figure 5.37 presents the options available for this purpose. The new options which are directly related to the racking system definition will now be described.

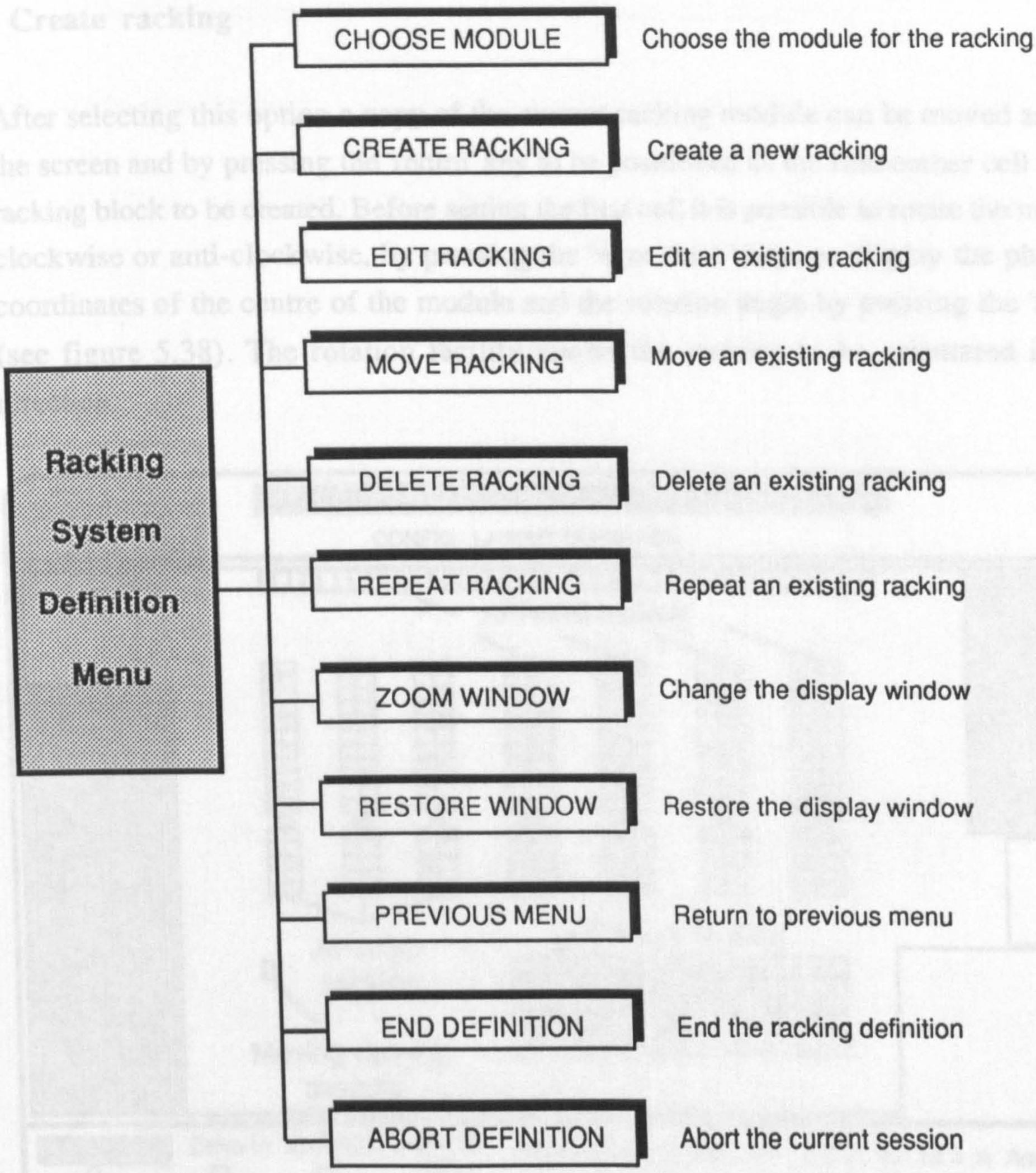


Figure 5.37 Racking system definition menu

. Choose module

With this option it is possible to select one of the existing racking modules, displayed in the command area (see figure 5.36), as the "current module". By default, the first created is automatically selected. The current module is used as the building block for the racking creation until another is chosen or the racking definition option is abandoned.

. Create racking

After selecting this option a copy of the current racking module can be moved around the screen and by pressing the 'return' key to be positioned as the first corner cell of the racking block to be created. Before setting the first cell it is possible to rotate the module clockwise or anti-clockwise, by pressing the '+' or the '-' key, or display the physical coordinates of the centre of the module and the rotation angle by pressing the 'P' key (see figure 5.38). The rotation facility allows the racking to be orientated in any direction.

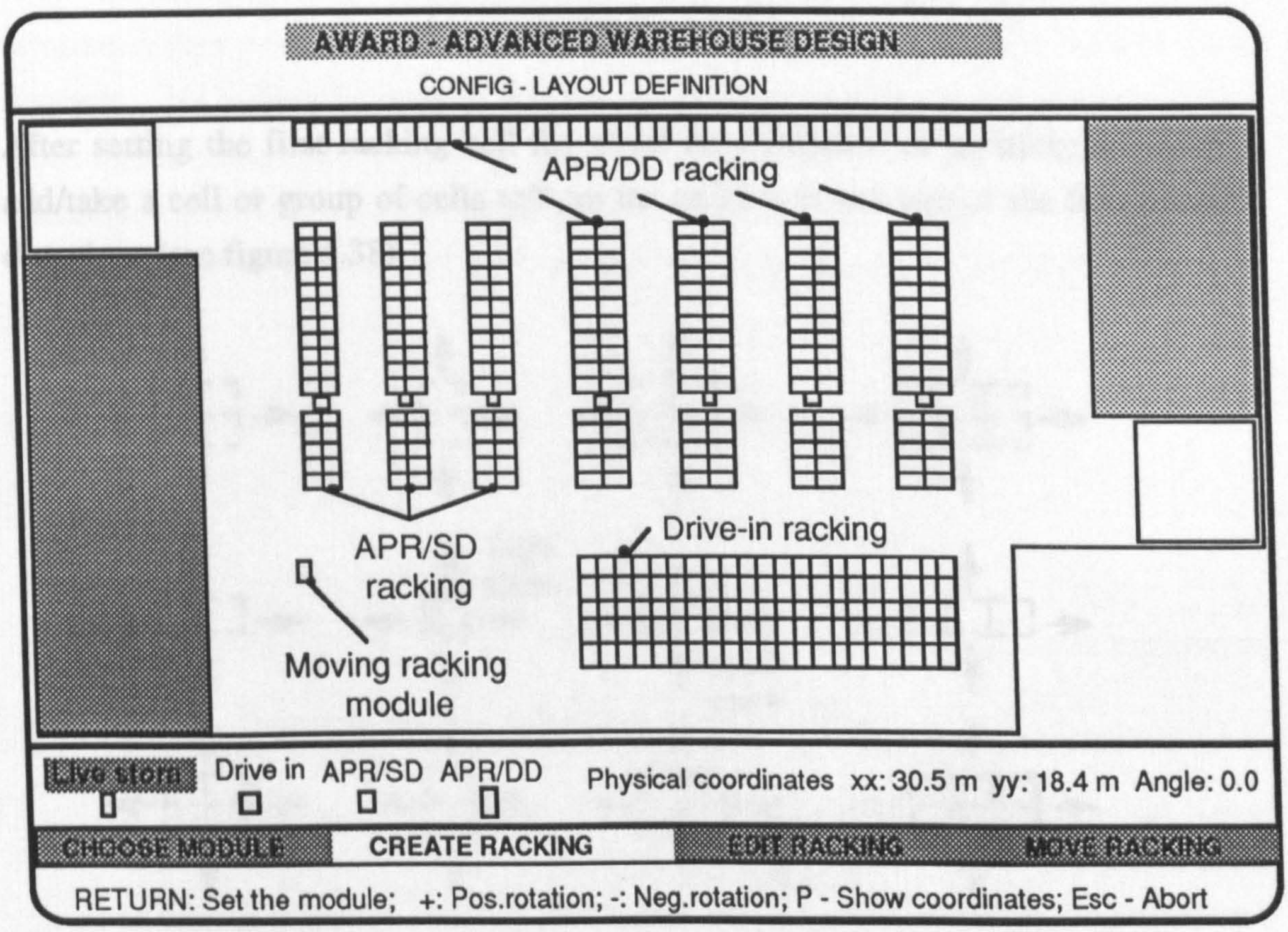


Figure 5.38 The warehouse layout showing the racking creation phase

During the first stage of the racking creation the following menu is displayed at the bottom of the command area (see figure 5.38):

RETURN - Set module: set the racking module, at the current position, as the first cell of the racking;

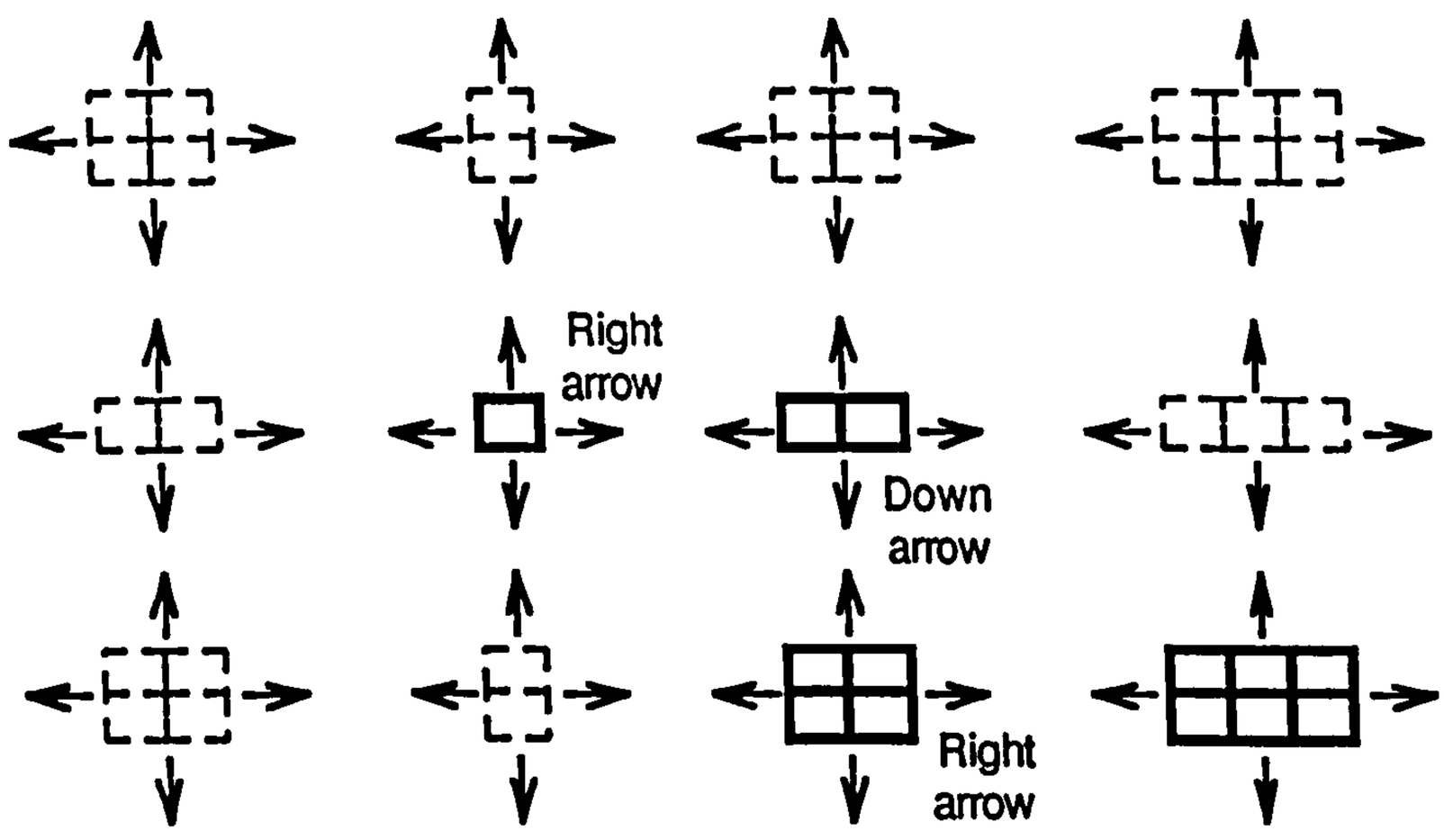
+: Positive rotation: rotate the module anti-clockwise;

-: Negative rotation: rotate the module clockwise;

P - Show coordinates: display the warehouse coordinates (metres), corresponding to the position of the centre of the racking module, and also the racking module rotation angle ;

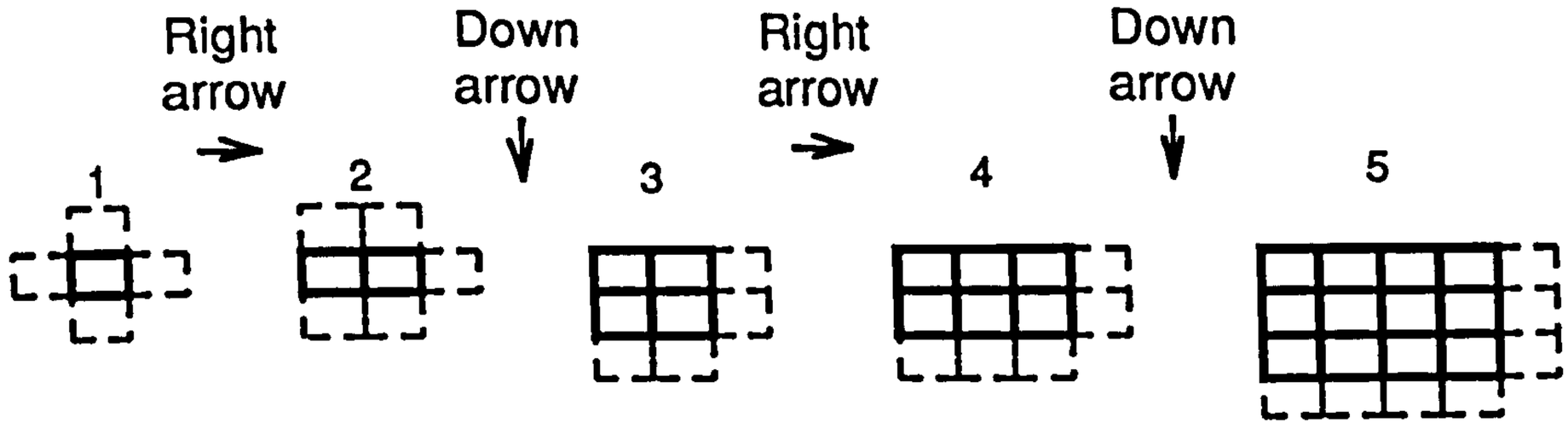
Esc - Abort: escape from the create racking option without creating the racking.

After setting the first racking cell the arrow keys (joydisk or joystick) are used to add/take a cell or group of cells to/from the racking in any one of the four possible directions (see figure 5.38).



5.39 Path in building a racking showing all possible arrow movements

Each time an arrow key is pressed, the racking will increase in that direction by repeating the row of cells, from the racking, in accordance with the arrow direction. At any time the last action can be reversed by pressing the arrow key in the opposite direction. Figure 5.39 shows different steps in building a racking using the arrow keys. The dashed lines represent possible evolutions from each stage.



5.40 Steps in building a racking using the arrow keys

When the racking has the required size pressing the 'return' key will end the racking creation. Other relevant information defined during this stage is the racking floor numbers. This is done by entering the first and last racking floor numbers into the data fields that are displayed, at the top of the command area, when the 'F' key is pressed. Although the default value for the first floor is one, and it is usually the case, there are situations where it is necessary to define a block of racking with no usable cells on the first floors.

During this second stage of racking creation the following menu is displayed:

Arrows:	create the racking as described;
RETURN: Quit	the racking remains as defined currently and control returns to the 'racking definition' menu;
F: Floor numbers	a two data fields screen is displayed, at the top of the command area, and the first and last racking floor numbers can be defined or modified;
Esc - Abort:	escape from the create racking option without creating the racking.

At the two racking creation stages extensive checking is made to prevent invalid racking layout. So, the initial racking module can not be set outside or intersecting the warehouse boundaries neither inside or intersecting a no-go area or another racking. During the second stage, the racking is only allowed to increase, in any direction, until it reaches an obstacle. When that happens the terminal bell bleeps and the racking size

remains the same. Furthermore, depending on the racking type the number of cells in one direction can be restrained. That is the case of the APR/SD and APR/DD where just one or two modules are acceptable, along the access direction, corresponding to one single row or two rows laid back-to-back. The access direction (direction used for pallet input/output to/from the racking) is always defined as the yy direction when the racking module is created (see figure 5.41).

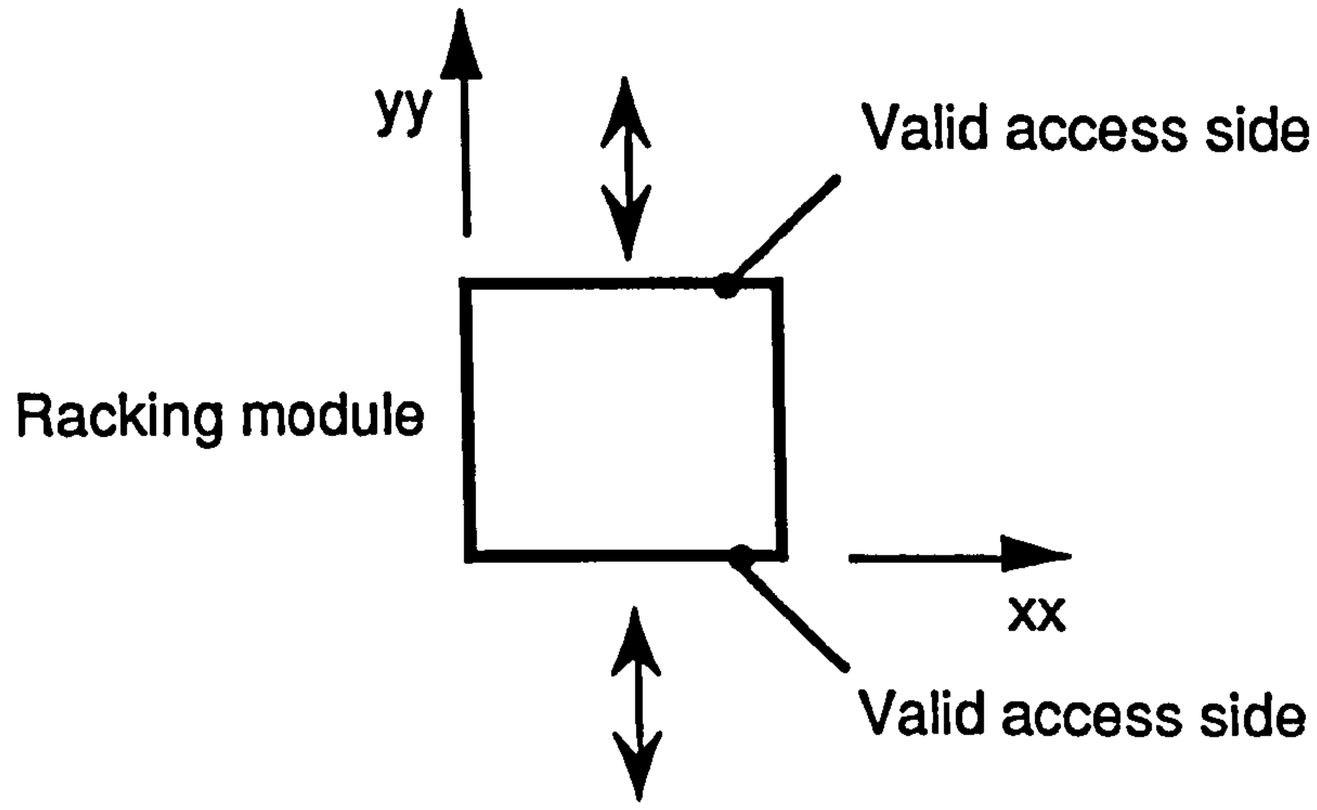


Figure 5.41 The racking module access direction

The racking module rotation angle defines the racking orientation and access direction. This angle is defined in relation to its initial position, as shown in figure 5.42, and the maximum permissible value is 90° both in clockwise and anti-clockwise directions.

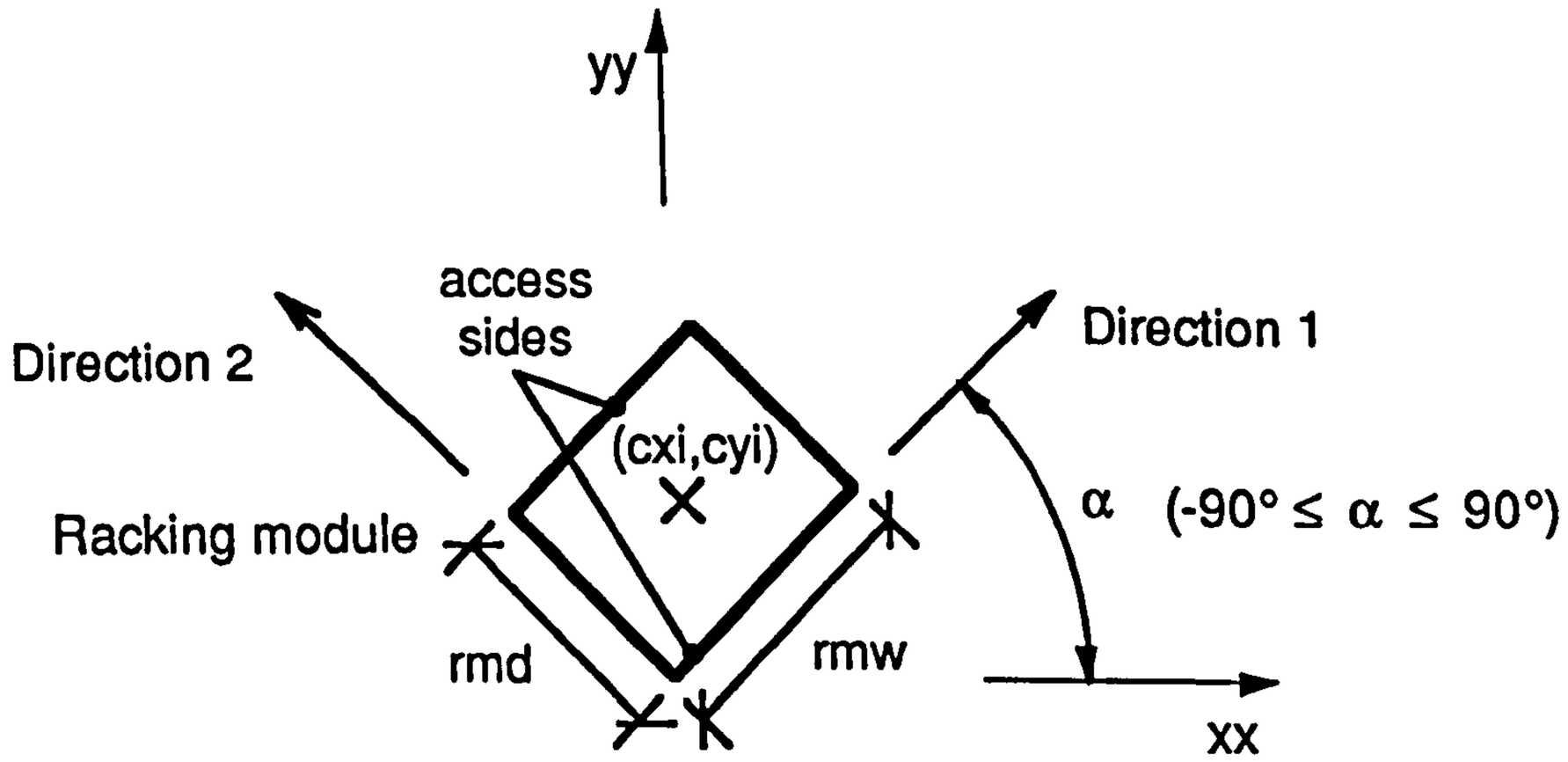


Figure 5.42 Racking module parameters definition

As by default the access direction is along the yy direction, it is necessary to rotate the racking module if the access is going to be made along any other direction. In figure 5.38, for the creation of the APR/SD racking, the racking module had to be rotated by 90° because the access is made along the xx direction.

For manipulating the racking blocks the following information is needed:

α	-	racking block rotation angle;
rmw	-	racking module width;
rmd	-	racking module depth;
nm1	-	number of racking modules along direction 1;
nm2	-	number of racking modules along direction 2;
cxi, cyi	-	first racking module centre coordinates;
df1, df2	-	direction 1 and 2 flags.

From the racking module rotation angle α, it is possible to obtain the direction cosines for direction 1 and 2 (see figure 5.42) as follows:

$$\begin{aligned} dcx1 &= \cos\alpha; \\ dcy1 &= \sin\alpha; \\ dcx2 &= -\sin\alpha; \\ dcy2 &= \cos\alpha. \end{aligned} \qquad (5.32)$$

The direction flags df1 and df2 are used to keep track of the way the racking was built. So, if the racking module was repeated along direction 1 then df1 would be equal to 1, if not, df1 would be equal to -1. The same principle applies to df2 in relation to direction 2. During the racking creation (represented in figures 5.39 and 5.40), everytime there is a change, it is necessary to know the racking overall dimensions, for checking purposes, and the coordinates of the changed modules for drawing or erasing purposes. The same algorithm can be used for both directions 1 and 2 if the right parameters are used. So, general directions **u** and **v** will be used and then, in each case, the correct correspondence will be made with directions 1 and 2.

The racking overall dimensions can be easily calculated by the following expressions:

$$lenu = nmu \,.\, widu$$
$$lenv = nmv \,.\, widv \qquad (5.33)$$

where lenu, lenv are the racking length along **u** and **v** directions;
nmu, nmv are the number of modules along **u** and **v** directions;
widu, widv are the module width along **u** and **v** directions.

The racking corner coordinates can be obtained by using the racking module centre coordinates, the direction cosines and the racking dimensions. The corner coordinates from the first module defined are equal to:

$$x_1 = cxi - \frac{lenu}{2}\, dcxu - \frac{lenv}{2}\, dcxv$$
$$y_1 = cyi - \frac{lenu}{2}\, dcyu - \frac{lenv}{2}\, dcyv \qquad (5.34)$$

where dcxu, dcyu are the direction cosines from **u** direction;
dcxv, dcyv are the direction cosines from **v** direction,

the other racking block corner coordinates can now be calculated as follows:

$$x_2 = x_1 + lenu \,.\, dcxu$$
$$y_2 = y_1 + lenu \,.\, dcyu$$

$$x_4 = x_1 + lenv \,.\, dcxv$$
$$y_4 = y_1 + lenv \,.\, dcyv \qquad (5.35)$$

$$x_3 = x_4 + lenu \,.\, dcxu$$
$$y_3 = y_4 + lenu \,.\, dcyu$$

These coordinates are used to check if the racking being created interferes with other elements. Some of this is done by using line intersection and inside polygon testing algorithms as described before. Also, as many of the elements used have rectangular shape, there is a need to find if two rectangles intersect each other. This is done by testing if each individual side of the first rectangle intersects any side of the second one. If this is not true, then it is enough to use one corner of each rectangle in turn, and

check if it is inside the other rectangle. This will test if any of the rectangles is inside the other one.

The algorithm that enables the racking creation, as described in figures 5.39 and 5.40, is based on a while loop that controls the input device and if that is the case defines the appropriate parameters and calls a subroutine that handles the racking modifications (see figure 5.43).

```
WHILE last key pressed is not "RETURN" or "ESC"
    Check input devices and return if an event occurs
    IF the right arrow key was pressed or equivalent THEN
        dcxu = dcx1; dcyu = dcy1; dcxv = dcx2; dcyv = dcy2 ;
        widu = rmw ; widv = rmd ; nmu = nm1 ; nmv = nm2; dfu = df1
        Call the subroutine to handle the racking modifications
    ELSE IF the left arrow key was pressed or equivalent THEN
        dcxu = dcx1; dcyu = dcy1; dcxv = dcx2; dcyv = dcy2 ;
        widu = rmw ; widv = rmd ; nmu = nm1 ; nmv = nm2; dfu = -df1
        Call the subroutine to handle the racking modifications
        df1 = -dfu
    ELSE IF the up arrow key was pressed or equivalent THEN
        IF df2 = 1 and nm2 = 2 and racking type is APR THEN
            Beep the terminal bell
        ELSE
            dcxu = dcx2; dcyu = dcy2 ; dcxv = dcx1; dcyv = dcy1 ;
            widu = rmd ; widv = rmw ; nmu = nm2 ; nmv = nm1; dfu = df2
            Call the subroutine to handle the racking modifications
        END IF
    ELSE IF the down arrow key was pressed or equivalent THEN
        IF df2 = -1 and nm2 = 2 and racking type is APR THEN
            Beep the terminal bell
        ELSE
            dcxu = dcx2; dcyu = dcy2 ; dcxv = dcx1; dcyv = dcy1 ;
            widu = rmd ; widv = rmw ; nmu = nm2 ; nmv = nm1; dfu = -df2
            Call the subroutine to handle the racking modifications
            df2 = -dfu
        END IF
```

```
    ELSE IF the "RETURN" key was pressed or equivalent THEN
        Increment the number of racking blocks
        Store all the relevant information for the racking block just created
    ELSE IF the key pressed or equivalent was different from "ESC" THEN
        Beep the terminal bell
    END IF
END WHILE
```

Figure 5.43 Algorithm to control the racking creation

The subroutine that handles the racking modifications is independent of the direction in which the racking is growing and the algorithm used is described in figure 5.44.

. Edit racking

This option allows the modification of some of the racking parameters. The parameters that can be changed are the number of modules and the floor numbers. The position of the initial racking module cannot be changed. If that becomes necessary then the 'move racking' option must be used to move the racking block as a whole. Before it is possible to do any changes, it is necessary to select the racking block to be edited. The selection is made by moving the graphic cursor into the desired racking block followed by pressing the 'return' key.

The selection algorithm is based on the inside polygon test. So, the graphic cursor must be inside the rectangular area, defined by the racking, for the selection to be successful. After having done this, the situation became similar to the second stage of the racking creation. The same menu is used as for editing the racking floor numbers or modifying the racking block as described in figures 5.39 and 5.40. The only difference concerns the action of the 'Esc' key which, if pressed, will make the control return to the 'racking definition' menu and all the changes will be disregarded restoring the racking block with its original data. The same level of checking, used during the racking creation, is used here to prevent the racking block from overlapping any of the existing elements.

```
IF dfu = 1 THEN
    Compute the new racking coordinates (using expressions 5.34 and 5.35)
    Check for interference with other elements
    IF there is interference THEN
        Beep the terminal bell
    ELSE
        cx = cxi + widu * nmu * dcxu                ! Compute the centre coordinates of
        cy = cyi + widu * nmu * dcyu                ! the first new module in direction u.
        nmu = nmu + 1                               ! Increment the number of modules.
        FOR i = 1 TO nm2
            cxa = cx + (i -1) * widv * dcxv         ! Compute the centre coordinates of
            cya = cy + (i -1) * widv * dcyv         ! each new module along direction v.
            Call the subroutine to draw the module with centre (cxa,cya)
        NEXT
    END IF
ELSE
    IF nmu = 1 THEN
        dfu = - dfu ; dcxu = -dcxu ; dcyu = -dcyu   ! Reverse direction
    ELSE
        nmu = nmu - 1                               ! Decrement the number of modules.
        cx = cxi + widu * nmu * dcxu                ! Compute the centre coordinates of
        cy = cyi + widu * nmu * dcyu                ! the last module along direction u.
        FOR i = 1 TO nm2
            cxa = cx + (i -1) * widv * dcxv         ! Compute the centre coordinates of
            cya = cy + (i -1) * widv * dcyv         ! each module along direction v.
            Call the subroutine to erase the module with centre (cxa,cya)
        NEXT
        IF nmu = 1 THEN
            dfu = - dfu ; dcxu = -dcxu ; dcyu = -dcyu   ! Reverse direction
        END IF
    END IF
END IF
```

Figure 5.44 Algorithm to handle the racking modifications

. Move racking

Any existing racking block can be moved to another valid position inside the warehouse. It is very important to be able to move the racking when testing different layouts. The racking selection procedure is identical to the one used in the 'edit racking' option. After selecting the racking block it can be moved, with the arrows (joydisk or joystick), to any place in the drawing area. Pressing the 'return' key will move the racking block to the new position, if valid, or will bleep the terminal bell, if it is in an invalid position, and nothing will happen. Pressing the 'Esc' key will escape from the 'move racking' option leaving the racking block in its original position.

. Delete racking

This option is used to delete unwanted racking blocks. The selection process is the same as the one used in the previous options. The selected racking block is then highlighted (blinking) and a question is displayed at the bottom of the screen to ask for the deletion to be confirmed. If the answer is 'yes' then the racking block is deleted, if not it is maintained. At anytime the 'Esc' key can be used to abandon the 'Delete racking' option without doing any deletion.

. Repeat racking

A warehouse racking system is usually made up of many identical blocks. In the example shown in figure 5.38 the APR/SD and the vertical APR/DD were obtained by repeating the first block created. So, it is very important and time saving to be able to duplicate a racking block any number of times. The selection process of the racking block to be repeated is the same as the one used in previous options. After the selection, a copy of the racking block can be moved around the screen, as in the 'move racking' option, and by pressing 'return' a new copy will be placed at the current position. The 'Esc' key is used here to abandon the 'repeat racking' option. Again the same level of checking is used, as in the previous options, to prevent the setting of racking block copies at invalid positions.

5.3.2.1.6. Transporter aisles definition

When running the simulation model a realistic representation is made of the transporter movement within the warehouse. For this to be possible it was necessary to find a solution that was easy to implement and had the desired level of accuracy. The solution found was to create a path network based on lines defining the centre of the transporter aisle (see figure 5.45).

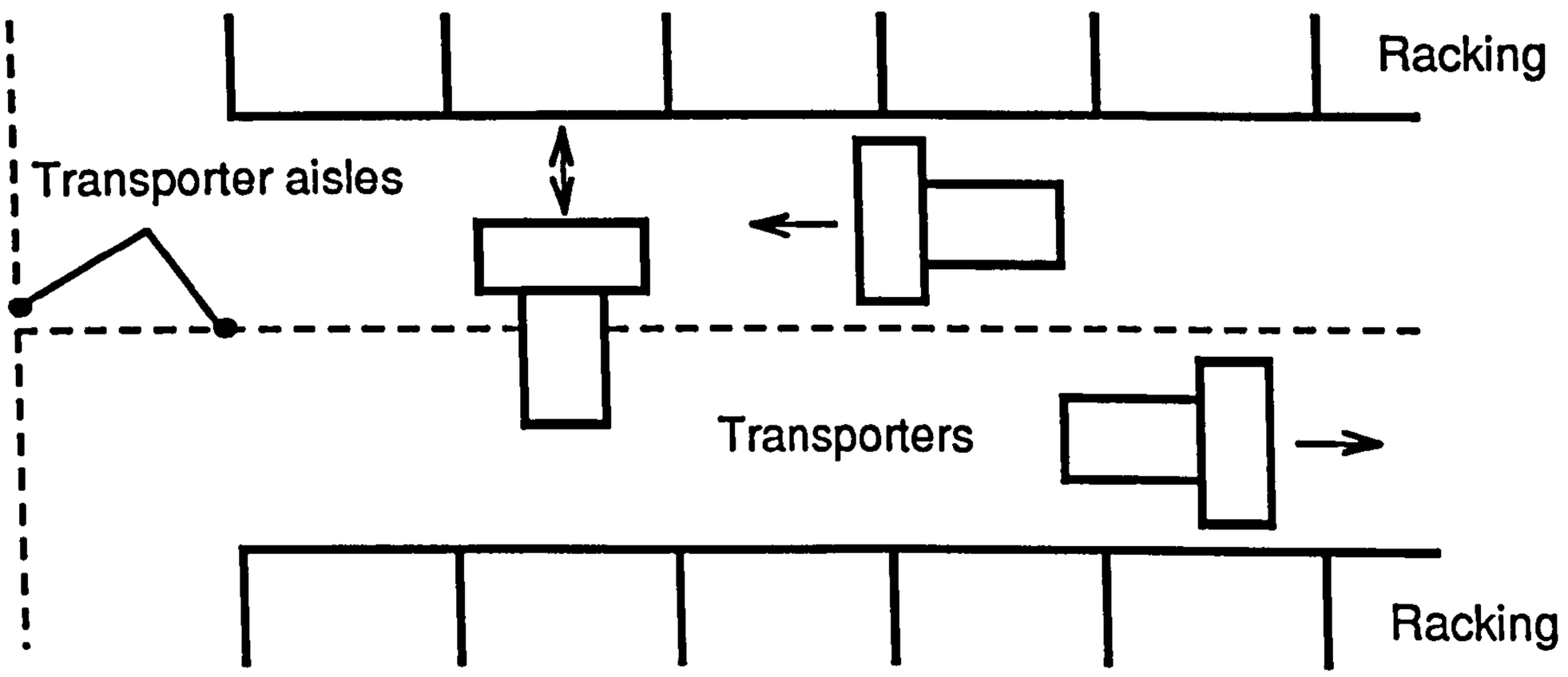


Figure 5.45 Transporter aisle showing different transporter positions

The transporter movement is made on opposite sides of the aisle (see figure 5.45) or along the centre of the aisle, depending on the aisle type and on the transporter type. The following two aisles types are considered:

- Normal aisle - this is the default aisle type; the transporters move on opposite sides of the aisle depending on the direction they take (see figure 5.45);

- Narrow aisle - the logic depends on the transporter type; if the transporter type is narrow aisle then just one transporter is allowed to get into the aisle and it moves along the centre of the aisle; if the transporter type is not narrow aisle then the aisle behaves as a normal aisle.

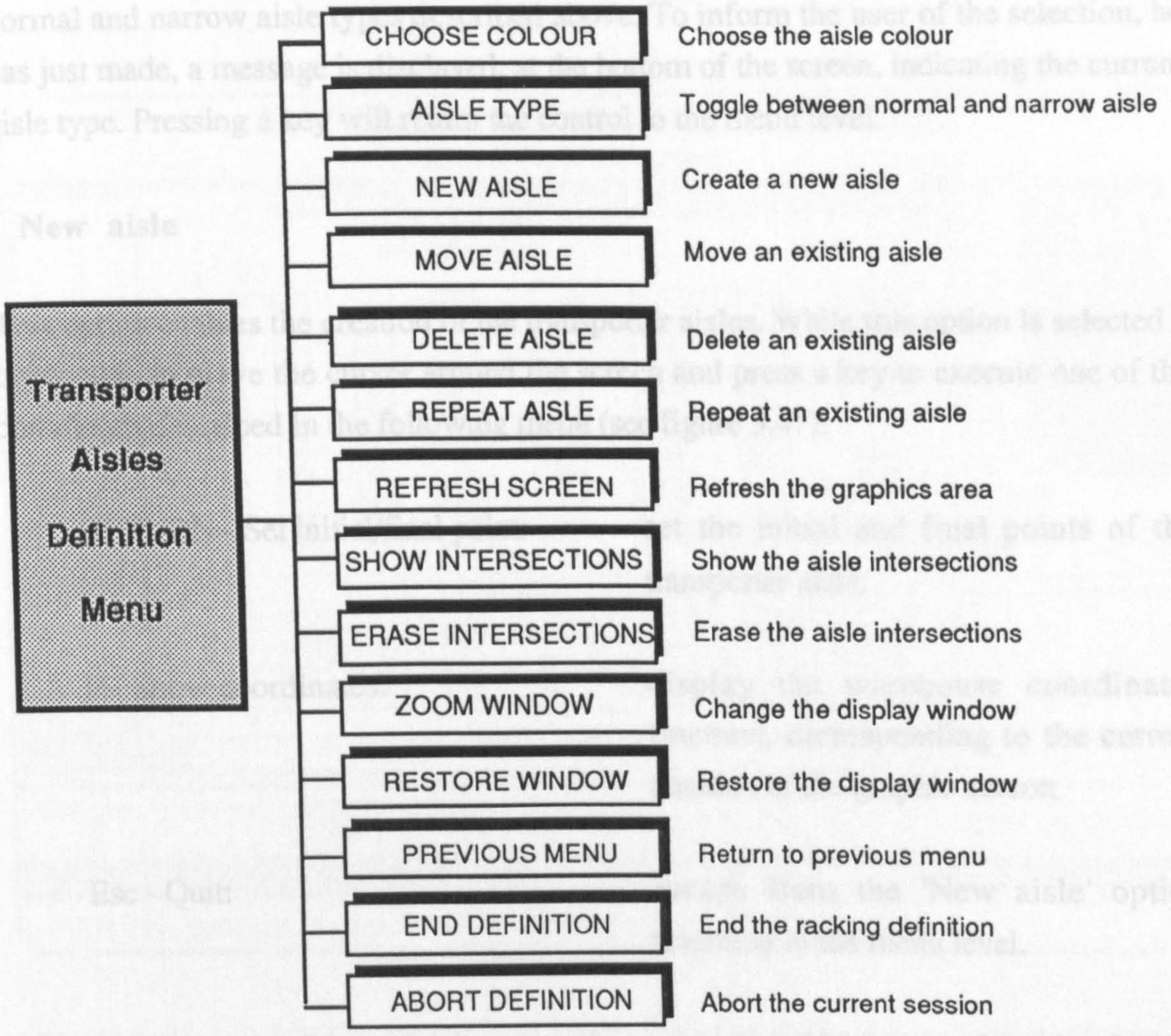


Figure 5.46 Transporter aisles definition menu

The transporters path network is used to define the specific path whenever it is necessary to move a transporter from one position to another within the warehouse. From all possible alternatives is chosen the one which has the shortest distance. Furthermore, the defined path is used to move the transporter during the simulation run.

All the options available during this stage are listed in figure 5.46 and the most important ones will be described next.

. Aisle type

This option is used to change the default aisle type for subsequently created aisles. Just by selecting the option the aisle type is toggled between the two available aisle types, the

normal and narrow aisle types described above. To inform the user of the selection, he has just made, a message is displayed, at the bottom of the screen, indicating the current aisle type. Pressing a key will return the control to the menu level.

. New aisle

This option enables the creation of the transporter aisles. While this option is selected it is possible to move the cursor around the screen and press a key to execute one of the commands described in the following menu (see figure 5.47):

RETURN - Set initial/final point:	set the initial and final points of the transporter aisle;
P - Show coordinates:	display the warehouse coordinates (metres), corresponding to the current position of the graphic cursor;
Esc - Quit:	escape from the 'New aisle' option returning to the menu level.

After selecting the 'New aisle' option the cursor should be moved to the desired transporter aisle initial position and the 'return' key pressed to set the origin of the aisle. Afterwards, the cursor should be moved to the desired transporter aisle end-position and the 'return' key pressed again to set the aisle. A rubber-banding facility is used to help position the aisle and the 'P' key can be pressed at any moment to get the current cursor physical coordinates. The aisle type and colour used in the aisle creation, are currently defined as the default ones. As in previous stages an on-line error check is made to prevent the intersection of the new aisle with existing warehouse elements.

. Move aisle, Delete aisle and Repeat aisle

All the these options share a common start which consist of the aisle selection. This is made by moving the graphic cursor to near the centre of the desired aisle followed by pressing the 'return' key. The selection algorithm used will select the aisle which has its centre at the shortest distance from the graphic cursor position. After selecting the aisle the way this option works is similar to the same options in previous stages e.g. in the racking creation.

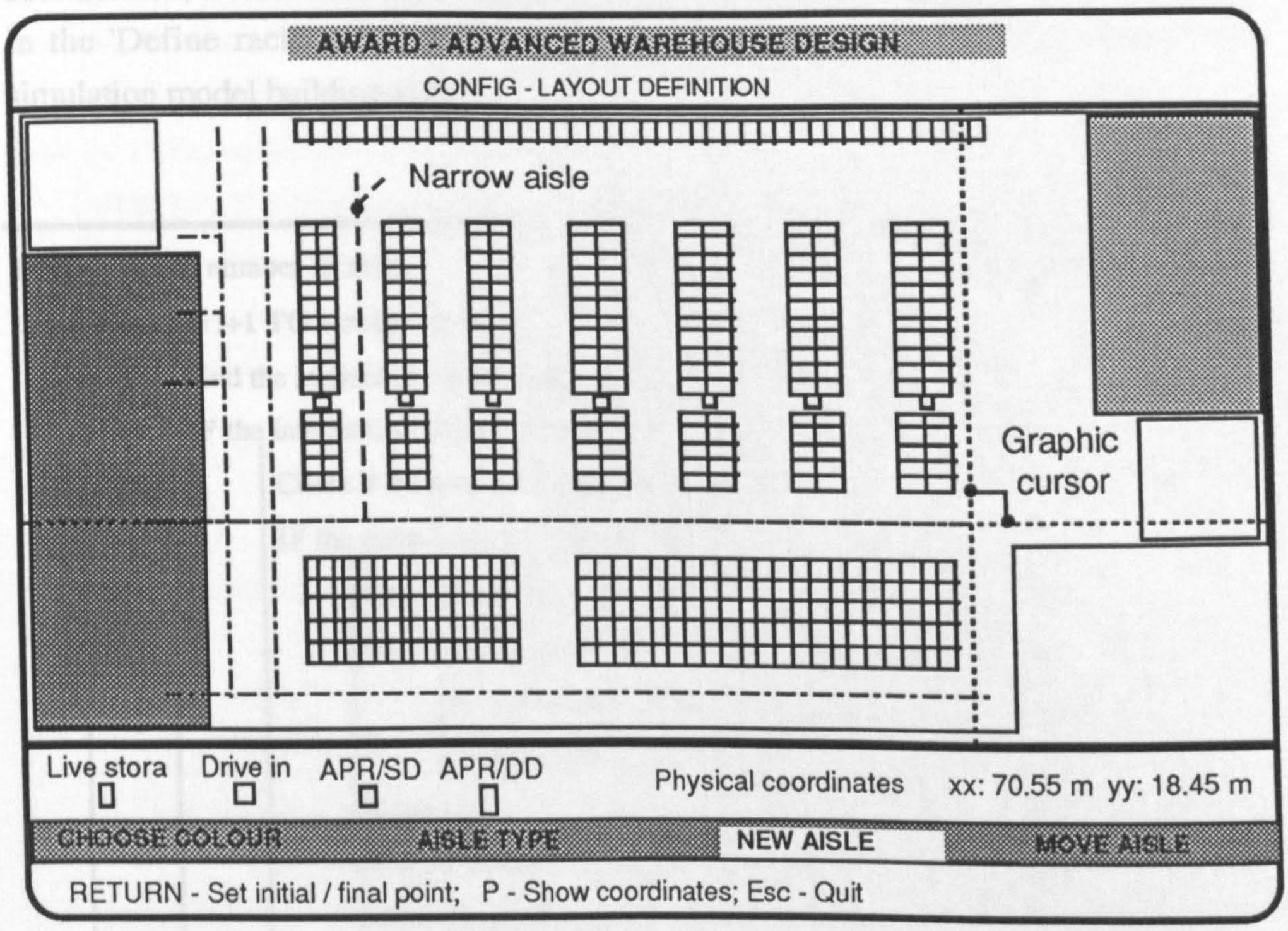


Figure 5.47 The warehouse layout showing the transporter aisles creation phase

. Show intersections

After defining the transporter path network it is necessary to make sure that all the junctions are identified. If that is not true the transporter movement will be affected in a way that they might not use the shortest path, between the origin and destination, or even will not be able to reach some areas within the warehouse. The only way to check the transporter path network is visually. In this option, all the branches and nodes, of the transporter path network, are identified and numbered. The branch and node numbers are then displayed on the screen so the user is able to check if the network is correct.

Figure 5.48 describes the algorithm used to identify all the nodes and branches in the transporter path network. This algorithm is used before displaying the branches and nodes numbers on the screen. Even if the option 'Show intersections' is not executed

this algorithm is automatically used, when the option 'Transporter aisles definition' is abandonned, to calculate the transporter path network data. This data is used, later on, in the 'Define racking access', in the 'Define transporters' options and also in the simulation model building stage.

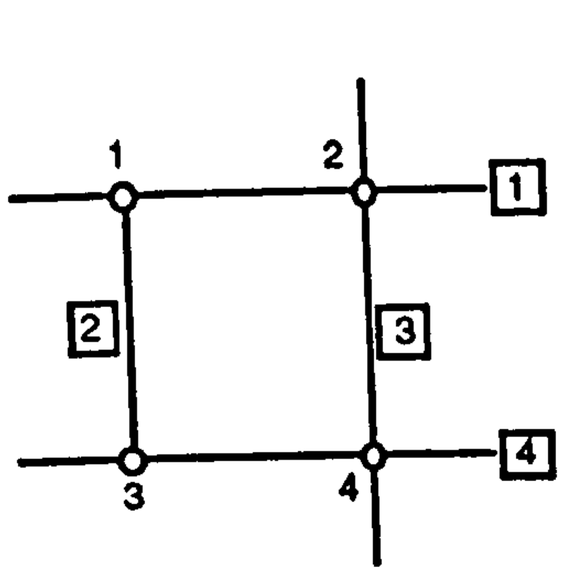
```
FOR i = 1 TO number of aisles                ! Identify all the aisle intersection and end points
    FOR j = i+1 TO number of aisles
        Find the intersection point between the lines containing aisles i and j
        IF the intersection point exists and lies on both aisles THEN
            Check if the intersection point is already identified
            IF the point is already identified THEN
                Check if the point is on the aisle i intersection points list
                IF the point isn't on the the aisle i intersection points list THEN
                    Increment the number of intersection points on aisle i
                    Add the point to the aisle i intersection points list
                END IF
                Check if the point is on the aisle j intersection points list
                IF the point isn't on the the aisle j intersection points list THEN
                    Increment the number of intersection points on aisle j
                    Add the point to the aisle j intersection points list
                END IF
            ELSE
                Increment the total number of nodes (intersection points)
                Store the xx and yy coordinates of the new node
                Increment the number of intersection points on aisles i and j
                Add the new node to the aisles i and j intersection points list
            END IF
        END IF
    NEXT
    Check if the aisle i end points are already on the intersection points list
    IF the aisle i end points aren't yet on the intersection points list THEN
        Update the total number of nodes (intersection points)
        Store the xx and yy coordinates of the new node(s)
        Update the number of intersection points on aisles i
        Add the new node(s) to the aisle i intersection points list
```

```
    END IF
NEXT
Renumber the nodes in ascending order from left to right and bottom to up of the screen
Sort in ascending order the nodes in the intersection points list for each aisle
FOR i = 1 TO number of aisles                                   ! Identify all the branches
    FOR j = 1 TO (number of intersections on aisle i) - 1
        Increment the number of branches
        Store the branch end points node numbers and aisle number
        Increment the number of branches on aisle i
        Store the branch number in the aisle i branch numbers list
    NEXT
NEXT
Renumber the branches in ascending order from left to right and bottom to up of the screen
```

Figure 5.48 Algorithm to identify the nodes and branches in the transporter path network

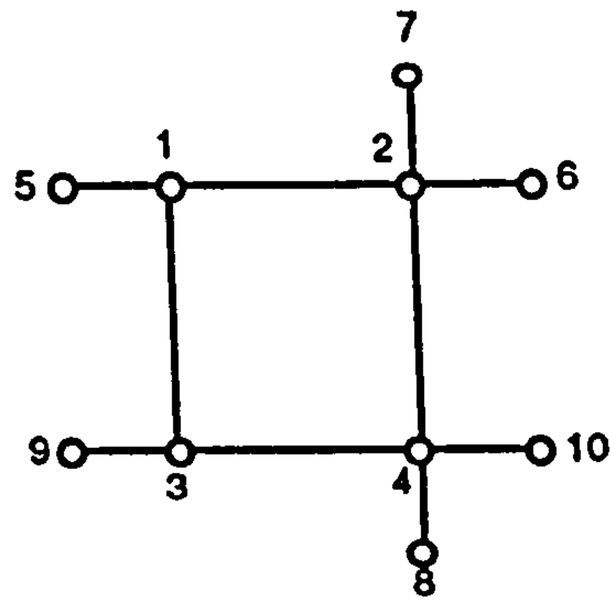
The algorithm described in figure 5.48 can be broken-down in to smaller steps that are easily understood. Those steps are presented in figure 5.49 through the use of a simple example. In this example four aisles are considered. The aisles are numbered in the same order as they are created. Although, in this example, only horizontal and vertical aisles were used it is allowable to create aisles orientated in any direction.

The first step, see figure 5.49 a), consists of finding all the aisle intersections. To each intersection point is given a sequential number and its xx and yy coordinates are stored. Associated with each aisle is stored the number of intersections and the intersection node numbers. In the second step, see figure 5.49 b), the aisle end points, that are not already identified, are numbered and the relevant data stored. The next step, see figure 5.49 c), will rearrange the data in a way that it will be easy to manipulate afterwards. First, the nodes are renumbered in ascending order from bottom to top and left to right.



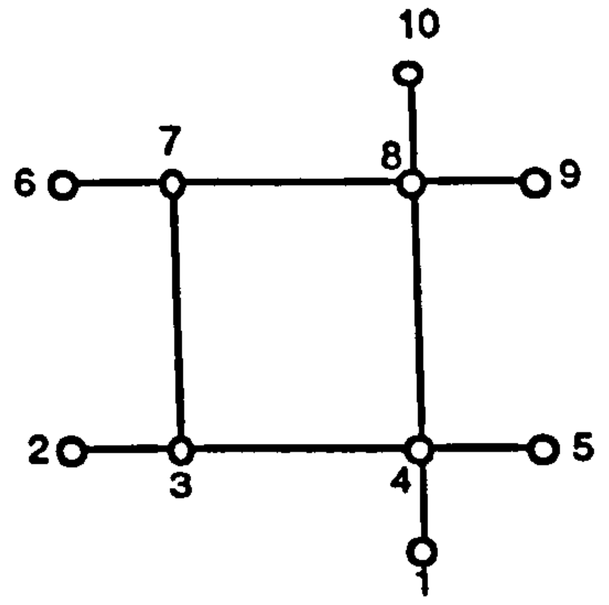
Aisle Nº	Intersection node numbers
1	1, 2
2	1, 3
3	2, 4
4	3, 4

a) Identify the intersections



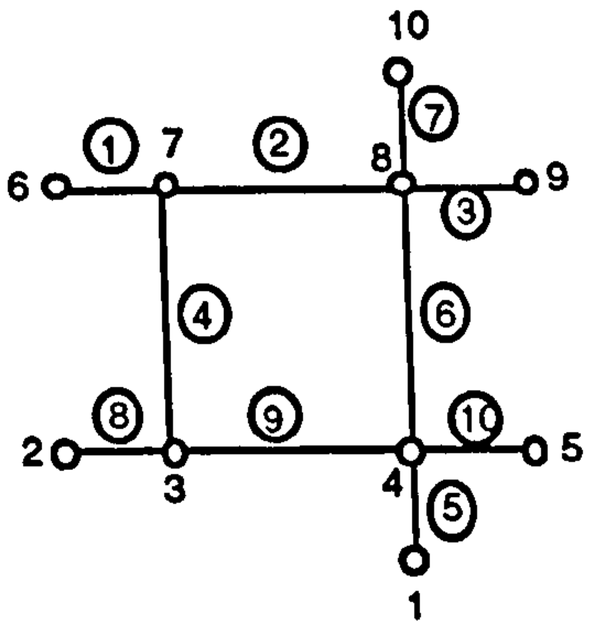
Aisle Nº	Intersection node numbers
1	1, 2, 5, 6
2	1, 3
3	2, 4, 7, 8
4	3, 4, 9, 10

b) Add the aisle end points



Aisle Nº	Intersection node numbers
1	6, 7, 8, 9
2	3, 7
3	1, 4, 8, 10
4	2, 3, 4, 5

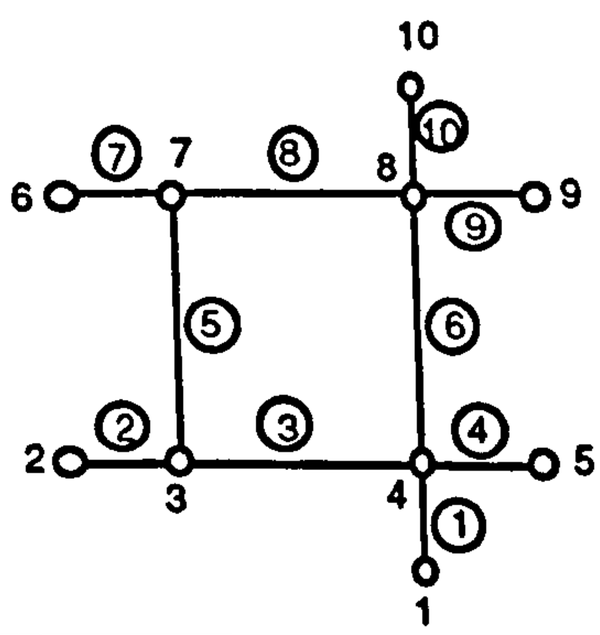
c) Renumber the nodes



Aisle Nº	Branch numbers
1	1, 2, 3
2	4
3	5, 6, 7
4	8, 9, 10

Branch Nº	Node numbers	Aisle Nº
1	6, 7	1
2	7, 8	1
3	8, 9	1
4	3, 7	2
5	1, 4	3
6	4, 8	3
7	8, 10	3
8	2, 3	4
9	3, 4	4
10	4, 5	4

d) Identify the branches



Aisle Nº	Branch numbers
1	7, 8, 9
2	5
3	1, 6, 10
4	2, 3, 4

Branch Nº	Node numbers	Aisle Nº
1	1, 4	3
2	2, 3	4
3	3, 4	4
4	4, 5	4
5	3, 7	2
6	4, 8	3
7	6, 7	1
8	7, 8	1
9	8, 9	1
10	8, 10	3

e) Renumber the branches

[1] - Aisle numbers; 1 - Node numbers; (1) - Branch numbers

Figure 5.49 Steps in identifying the nodes and branches of the transporter path network

Then, the intersection node numbers list, for each aisle, will be sorted so that the node numbers will correspond to consecutive intersection points along the aisle. After doing this, it is possible to identify all the branches in the transporter path network. Each pair of consecutive nodes, in each aisle, defines a new branch (see figure 5.49 d)). Finally, the branches are renumbered, see figure 5.49 e), from bottom to top and left to right using the branch middle point coordinates.

Renumbering the branches helps the visual checking of the transporter path network (see figure 5.50), the only way to assure that everything is correct. The intersection nodes are signalled by small square markers and the branch numbers are displayed near the middle of the aisle (see figure 5.50). All this information is kept on screen until the 'Transporter aisle definition' option is abandoned or the 'Erase intersections' option is executed. However, the information displayed is static so, if aisles are created, deleted or moved, it is necessary to execute the 'Show intersections' option again for the nodes and branches information to be updated on the screen.

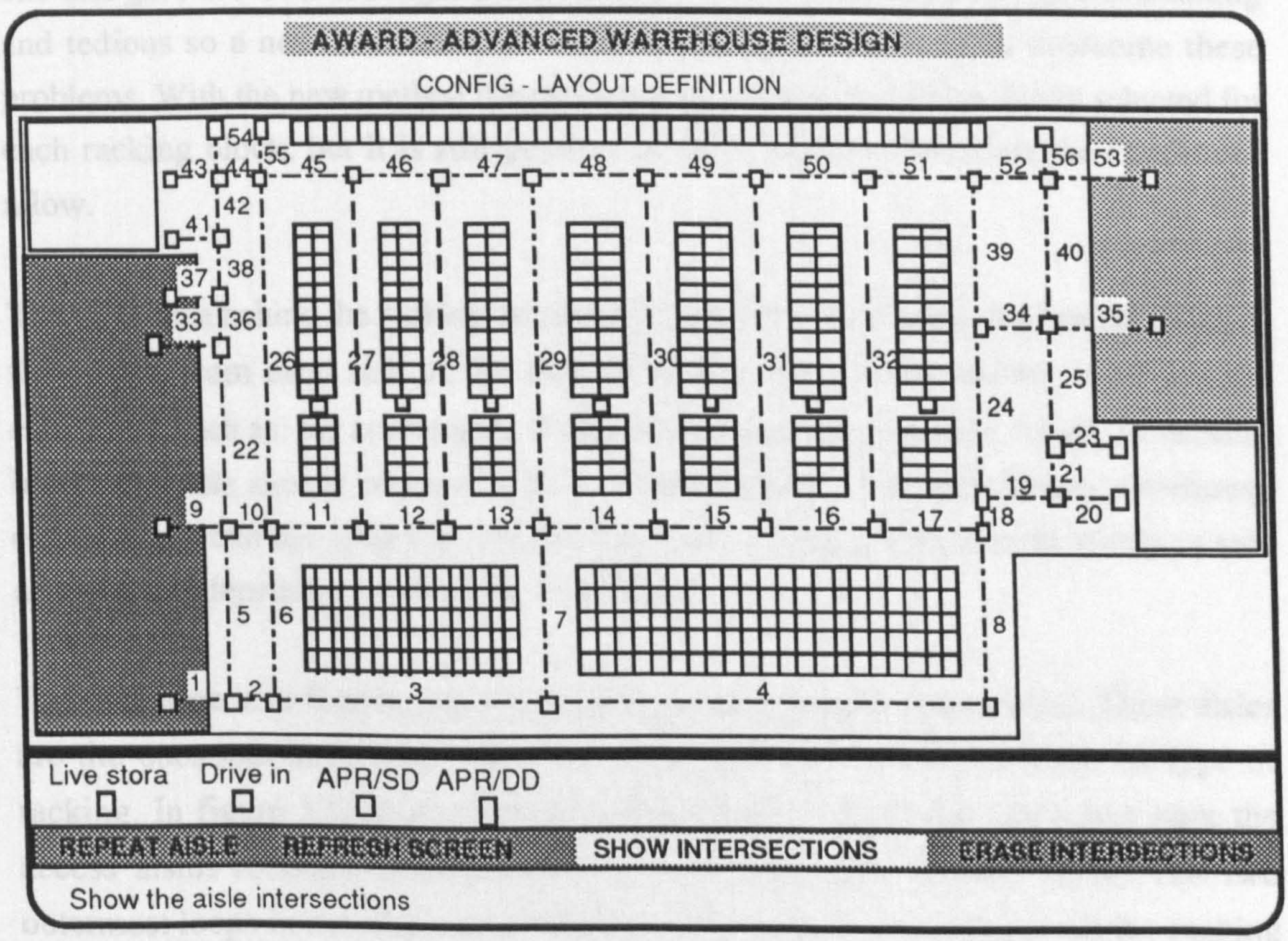


Figure 5.50 The warehouse layout showing the aisle intersections

. Erase intersections

The only objective of this option is to erase the node markers and branch numbers from the screen. The usefulness of it consists in clearing the screen from redundant information that can cause difficulties to the transporter path network design process.

5.3.2.1.7. Racking access definition

For the transporters to be able to take pallets to and from the racking, it is necessary, first of all, to know the aisle that gives access to each individual racking. So, for each racking block, must be defined the aisle that give access to it. In the first developed prototype, see 5.2.2., each racking block was highlighted in sequence and then it was necessary to go through the aisles, using the arrow keys or the joydisk, and select the one that gave access to the highlighted racking block. This process was time consuming and tedious so a new and automatic method has been developed to overcome these problems. With the new method the racking access aisles are automatically selected for each racking block, but it is still possible to make changes whenever the conditions allow.

The basic idea behind the racking access aisles algorithm is to select the nearest aisles, if they exist, from each side of the racking block. There are, however, some logical constraints such as: the access aisle should be the same, on each side, for all the racking block; the aisle should be parallel to the racking block; there can't be any warehouse element between the aisle and the racking; each racking block must have one or two access aisles depending on the type of racking.

The first stage is to find for each racking block the potential access aisles. These aisles are the ones that satisfy the above contraints except in what concerns the type of racking. In figure 5.51 is described the algorithm to find those aisles and store the access aisles relevant information associated with each racking block. The two outermost loops in the algorithm of figure 5.51 are used to go through all the racking block and for each one to consider each racking end in turn. The racking ends are the lines segment in the end of the racking in a plan view (see figure 5.52). The innermost loop, in the figure 5.51 algorithm, is used to step through each transporter aisle. This

enables one to find the intersection between each transporter aisle and the line containing the racking end (see figure 5.52).

```
FOR each racking block
    Call the subroutine to calculate the racking block corner coordinates
    FOR each racking end
        FOR each transporter aisle
            Find the intersection between the aisle and the line containning the racking end
            IF the intersection point exists THEN
                Compute the distance between the point and both racking end corners
                IF the bottom corner is at the shortest distance THEN
                    IF current distance is smaller then previous ones (bottom) THEN
                        Update the minimum distance (bottom)
                        Store the current aisle number (bottom)
                        Store the intersection point coordinates (bottom)
                    END IF
                ELSE
                    IF current distance is smaller then previous ones (top) THEN
                        Update the minimum distance (top)
                        Store the current aisle number (top)
                        Store the intersection point coordinates (top)
                    END IF
                END IF
            END IF
        NEXT
        IF it is the first racking end THEN
            Store for the current racking block the bottom and top access aisle numbers
        ELSE
            Check if the access aisle numbers are the same at both racking ends
            IF they aren't the same for the bottom access THEN
                Signal an error for the current racking block bottom access
            END IF
            IF they aren't the same for the top access THEN
                Signal an error for the current racking block top access
            END IF
        END IF
```

```
    NEXT
    Check if there is any element between the bottom and top access aisles and the racking block
NEXT
```

Figure 5.51 Algorithm to automatically detect the potential racking block access aisles

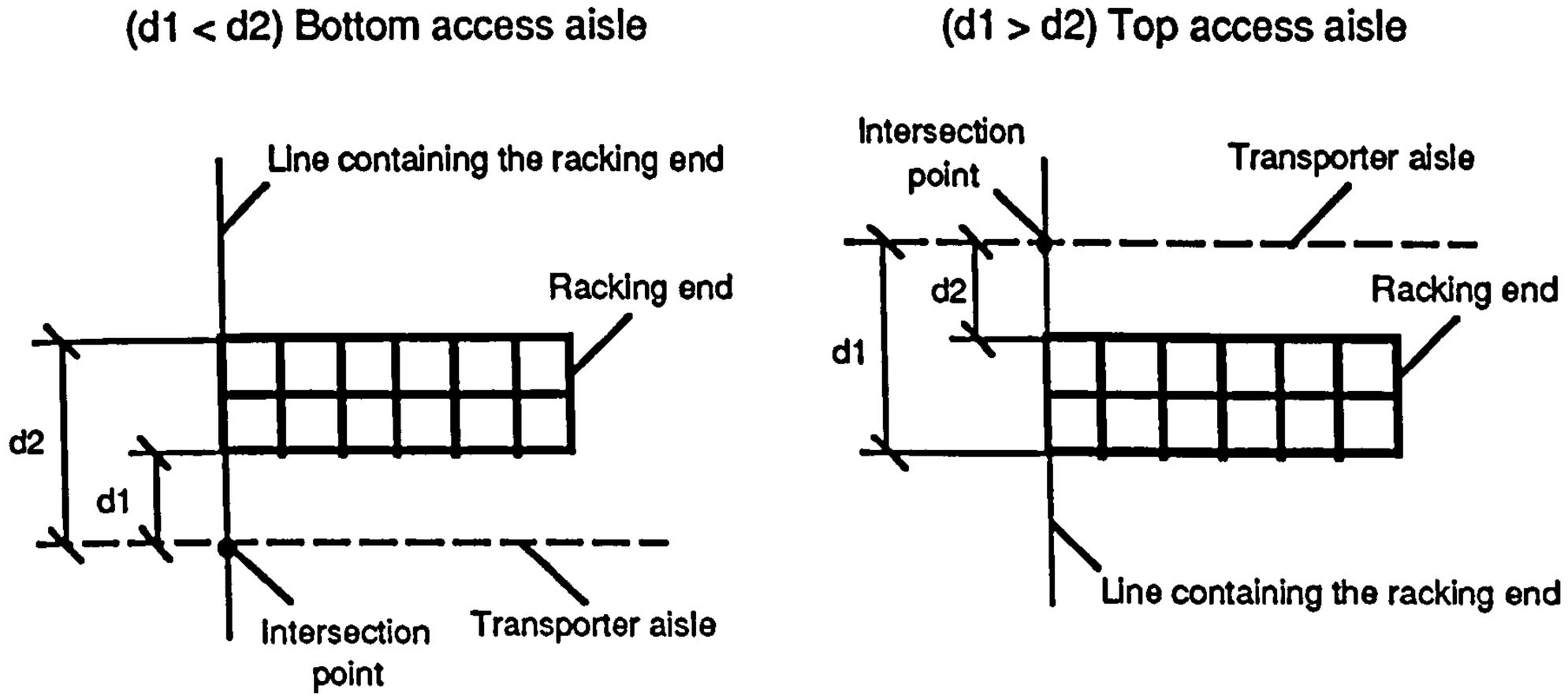


Figure 5.52 Parameters used in the algorithm to detect the racking block access aisles

The line containing the racking end is defined by the two racking end corner coordinates. The Racking block coordinates are calculated, in a separate module, using expressions (5.34) and (5.35) (see 5.3.2.1.5.). After finding the intersection point, if it exists, it is possible to calculate the distances d1 and d2 (see figure 5.52) between the intersection point and the bottom and top corners of the racking end. Comparing the values from d1 and d2 it determines if the current aisle is a potential bottom or top access aisle (see figure 5.52). The objective is then to find the nearest aisle from the bottom and the top of the racking block. This is done by selecting the aisle for which distance d1 is minimum, for the bottom access aisle, and the aisle for which distance d2 is minimum for the top access aisle. This selection is done in sequence for both ends of the racking block.

After finishing the loop for all the transporter aisles (see figure 5.51) and if it is the first racking end then it is stored, associated with the current racking block, the relevant information about the access aisles. If it is the second racking end then it is only

checked if the bottom and top access aisles are the same as the ones found for the first racking end. If they are not the same then it is signalled as an internal error. Before stepping to the next racking block it is also checked if there are any warehouse elements between the access aisles and the racking block. If there are, it is signalled as an internal error.

After detecting all the racking block potential aisles another algorithm is used to define all the possible access alternatives depending on the type of racking. In figure 5.53 is represented the different access situations for a single row of APR/SD or APR/DD (see 5.3.2.1.4.).

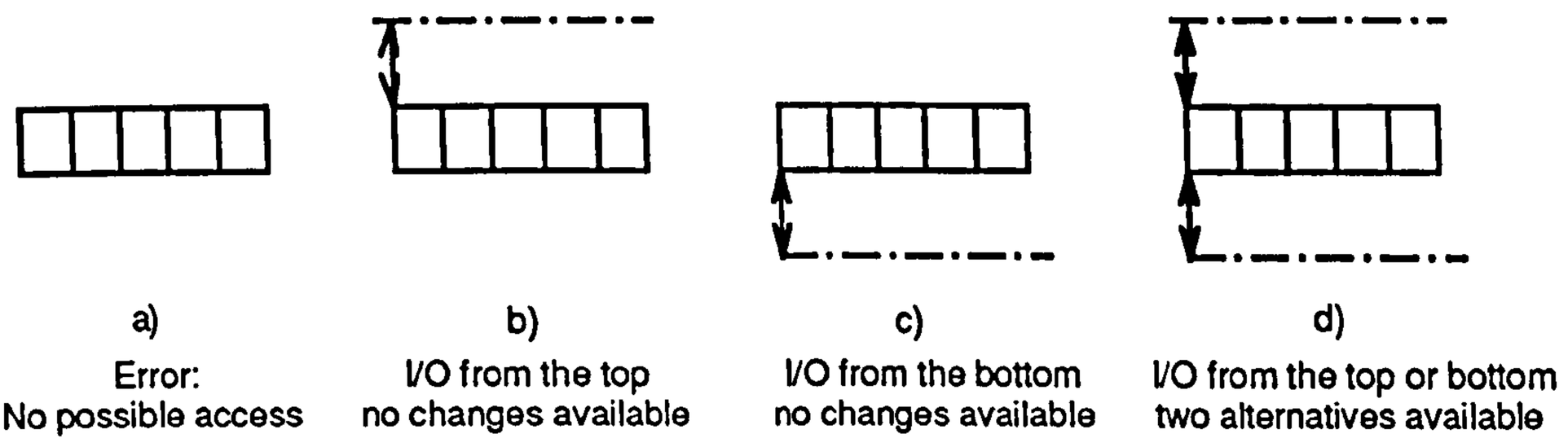


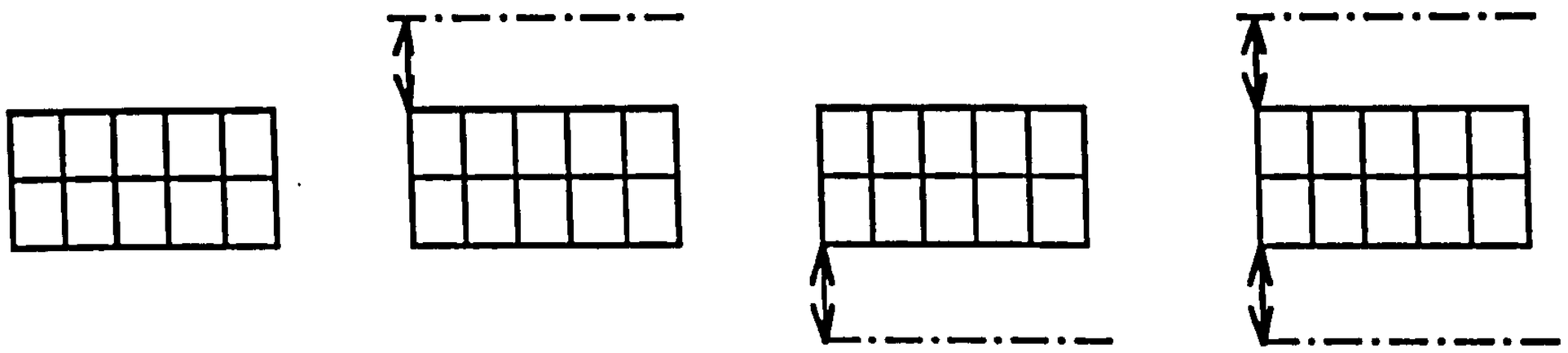
Figure 5.53 The different access situations for a single row of APR/SD or APR/DD

The situation described in figure 5.53 a) occurs when there are no potential access aisles available for the racking block. This can happen because the algorithm, from figure 5.51, was not able to find any access aisle or has detected an incorrect condition for the racking block access aisles. When entering the option 'Define the racking access' the racking blocks are scanned in sequence and if the situation of figure 5.53 a) arises an error message in displayed at the bottom of the screen and the racking block will blink to signal the error.

The situations described in figure 5.53 b) and c) are the most common when a single row of APR/SD or APR/DD is used. Usually the APR/SD or APR/DD are combined in two rows placed back to back. When only a single row is used normally it is placed near a wall (see figure 5.50) and the access can be made only from one side. In the cases b) and c), from figure 5.53, the access aisle is automatically defined and no changes can be made. The last situation, described in figure 5.53 d), although less common can happen if there are two valid access aisles on opposite sides of the racking

block. In this case, one is automatically selected, by default the bottom one, but it is possible to select any of the two alternatives represented in figure 5.53 d).

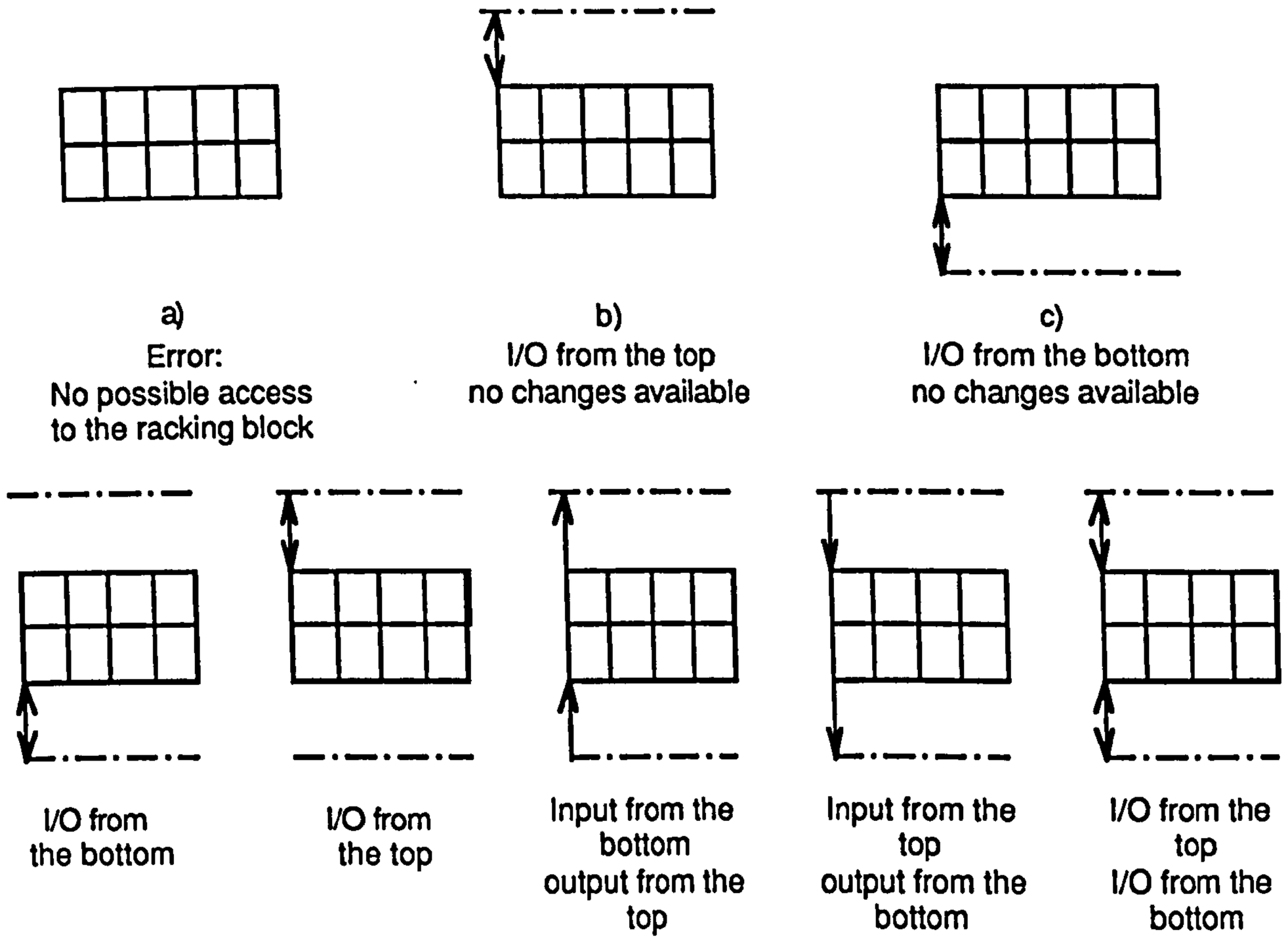
As it was said above, usually the APR/SD or the APR/DD are combined in two rows placed back to back. Figure 5.54 describes the different access situations that can appear in this case. The first three situations a), b) and c), from figure 5.54, correspond to error conditions. If there is no valid access aisles for one or both sides of the racking block then an error condition occurs and is taken a similar procedure to the one described for figure 5.53 a). The only possible valid situation is the one described in figure 5.54 d). For each side of the racking block is automatically selected one aisle. The pallet I/O is made to the same aisle on each side of the racking. In this case there are no alternatives available so, no changes can be made.



a)	b)	c)	d)
Error: No possible access to both racking sides	Error: No possible access to bottom racking side	Error: No possible access to top racking side	I/O from the top and I/O from the bottom No changes available

Figure 5.54 The different access situations for back to back APR/SD or APR/DD

In figure 5.55 is described the different access situations for the PMR, Drive-in or Block-stacking type of racking (see 5.3.2.1.4.). The three situations a), b) and c) from figure 5.55 are similar to the equivalent situations described in figure 5.53 for the single row APR/SD or APR/DD.



d) I/O from the top/bottom: five alternatives available

Figure 5.55 The different access situations for PMR, Drive-in or Block-stacking

If there are two potential access aisles, one in each side of the racking block, then there is a choice between five different alternatives described in figure 5.55 d). The most common ones are the first two where the pallet I/O is made from just one aisle.

In figure 5.56 is described the different access situations for the Live storage type of racking (see 5.3.2.1.4.). Because with this type of racking the input and output is always made from opposite sides of the racking there must be one access aisle on each side of the racking or an error occurs (see figure 5.56 a), b) and c)). The only valid situation is represented in figure 5.56 d). In this case, there is a choice between two alternatives: the pallet input is made from the bottom and the pallet output is made from the top (default); or the pallet input is made from the top and the pallet output is made from the bottom.

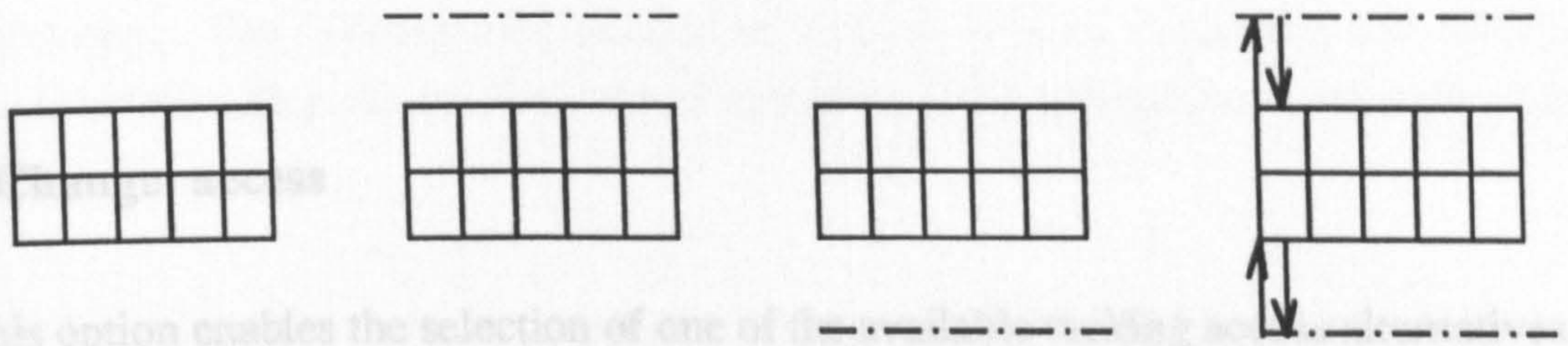


Figure 5.56 The different access situations for Live storage racking

After selecting the 'Racking access definition' option the racking block potential access aisles are defined, using the algorithm from figure 5.51, and the different access alternatives are found, depending on the type of racking, as described above. Also, unless an error situation occurs, one racking block access alternative is selected as default. In figure 5.57 are listed the available options for the 'Racking access definition'. Only the first option is directly related to the racking access definition having all the others been previously described. Next, we consider the 'Change access' option.

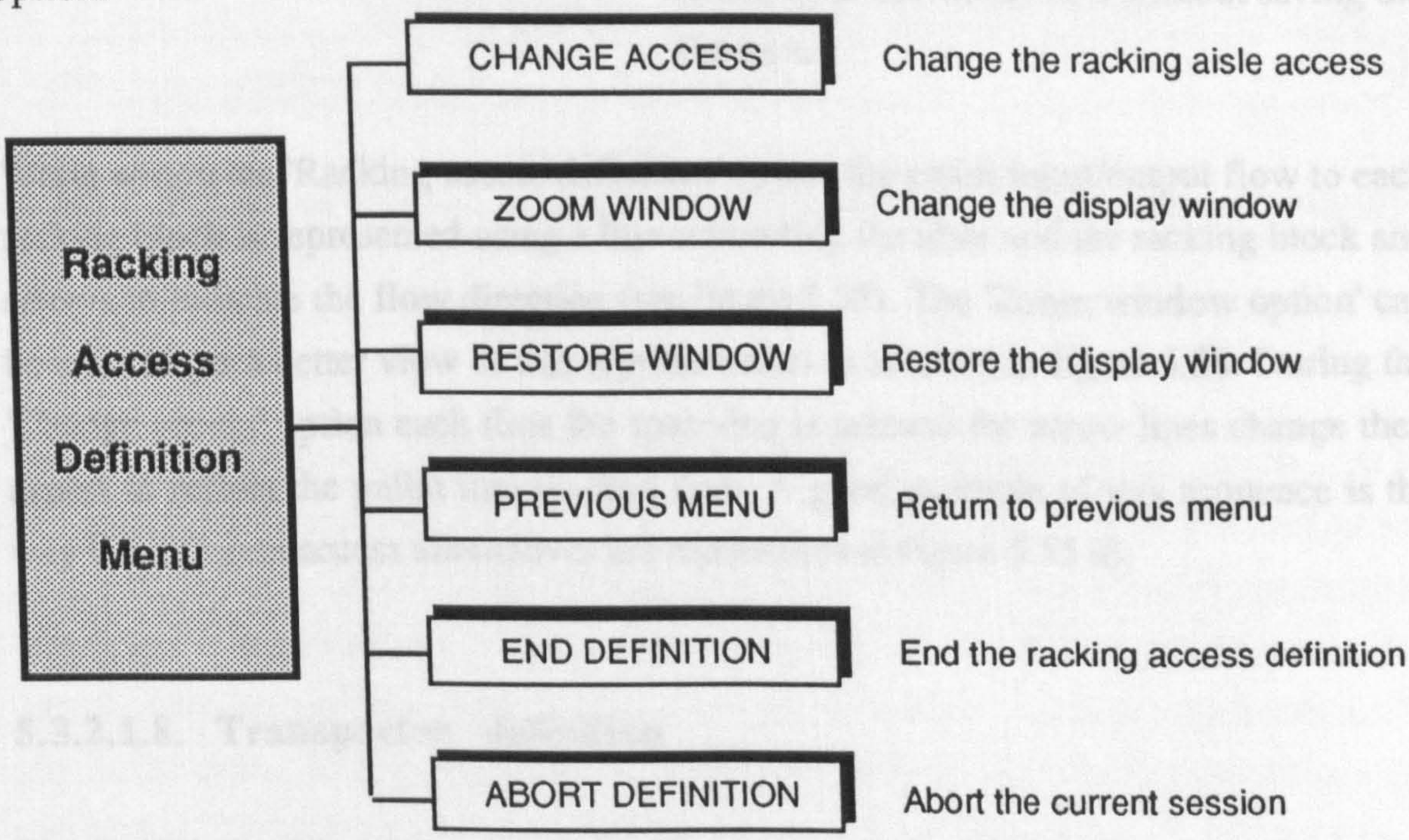


Figure 5.57 Racking access definition menu

. Change access

This option enables the selection of one of the available racking access alternatives for a racking block. The number of different alternatives, as described before, depends on the potential access aisles and on the racking type. After entering the 'Change access' option it is necessary to select the desired racking block. This is done by moving the cursor, with the joydisk or arrow keys, over the racking block and then by pressing the 'Return' key. If no changes are available a message is displayed at the bottom of the screen otherwise the following menu is displayed:

Space bar - Change access:	step through the different racking access alternatives;
RETURN - Set access:	Set the selected alternative as the current one and quit the 'Change access' option;
Esc - Quit:	escape from the 'Change access' option returning to the menu level without saving the changes.

While within the 'Racking access definition' option the pallet input/output flow to each racking block is represented using a line connecting the aisle and the racking block and arrows to indicate the flow direction (see figure 5.58). The 'Zoom window option' can be used to get a better view of this representation as showed in figure 5.58. During the 'Change access' option each time the space-bar is pressed the arrow lines change their aspect to reflect the pallet input/output flow. A good example of this sequence is the way the different access alternatives are represented in figure 5.55 d).

5.3.2.1.8. Transporter definition

The handling equipment is one of the key factors in a warehouse system. There is a large range of handling equipment available with various characteristics which are suitable for different type of storage solutions. The word transporter is adopted as the generic term to name any handling equipment used to move goods within the

warehouse. The 'Transporter definition' option is used to define the transporter characteristics (e.g. dimensions, travel speed etc.), the transporter movement path and their park position.

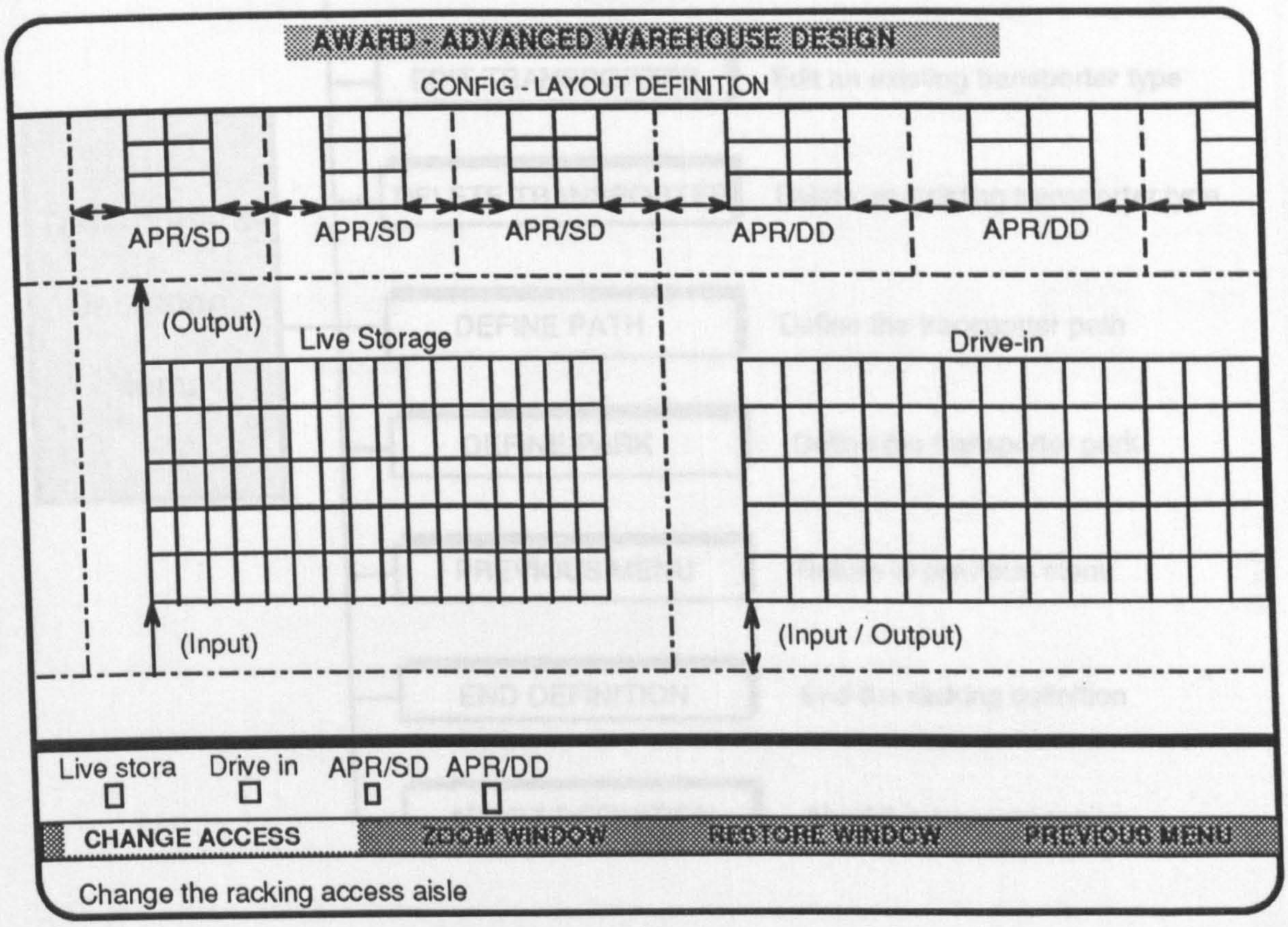


Figure 5.58 The warehouse layout showing the racking access aisles (zoom window)

The transporter movement path is defined by specifying the aisles which can be used by each particular transporter. This is useful because it allows one to dedicate transporters to different areas within the warehouse. The transporter park position is the location where the transporter should go to and stay parked while it is in an idle state. The potential transporter park positions are the aisle ends when they are not intersected by any other aisle. In Figure 5.59 are listed the available options for the transporter definition. The options directly related to the transporter definition will be considerer next.

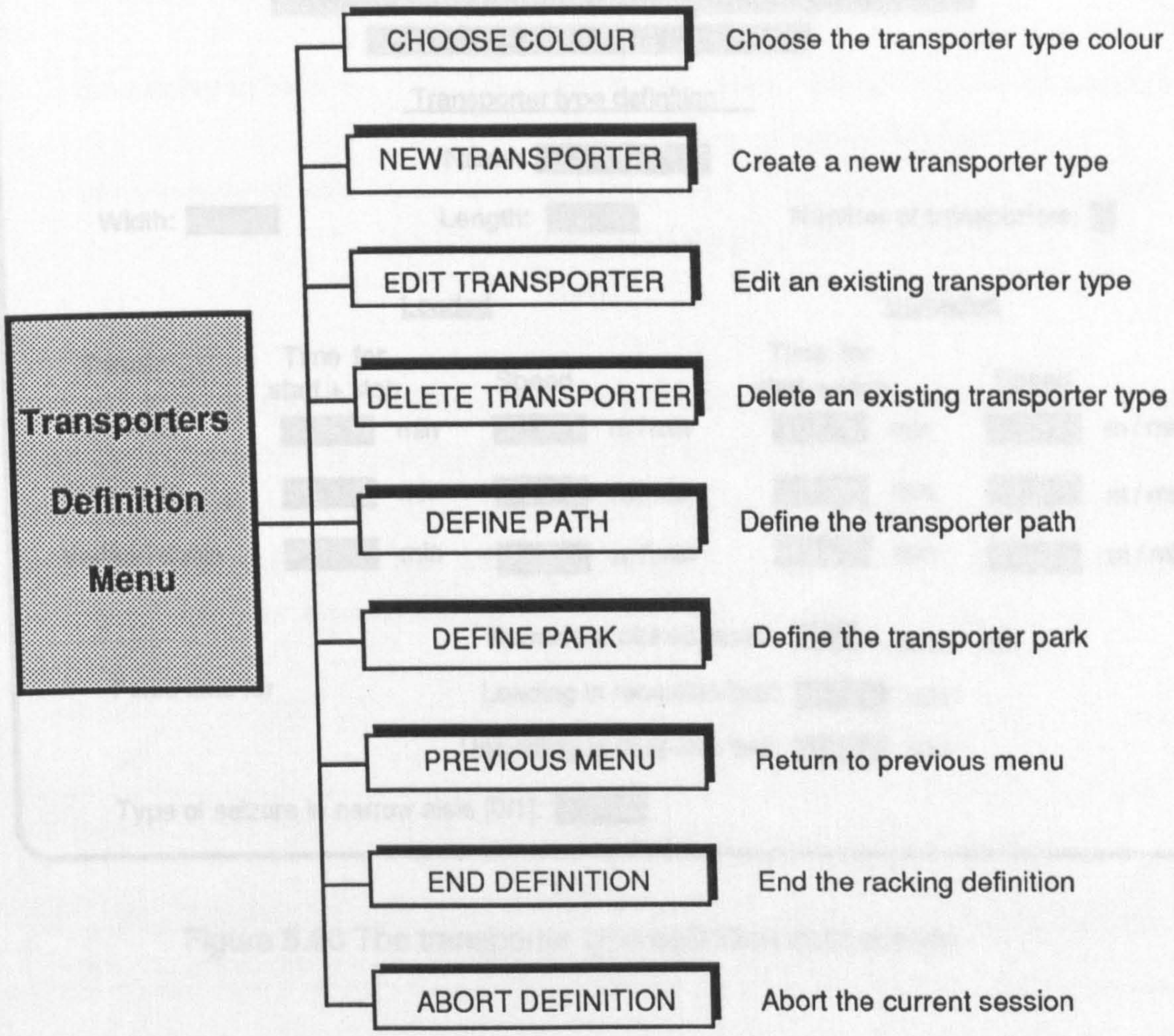


Figure 5.59 Transporters definition menu

. New transporter

With this option it is possible to create a new transporter type and define its characteristics. The transporter type is used to store the characteristics (colour, dimensions, speed etc.) from a set of transporters that share common characteristics.

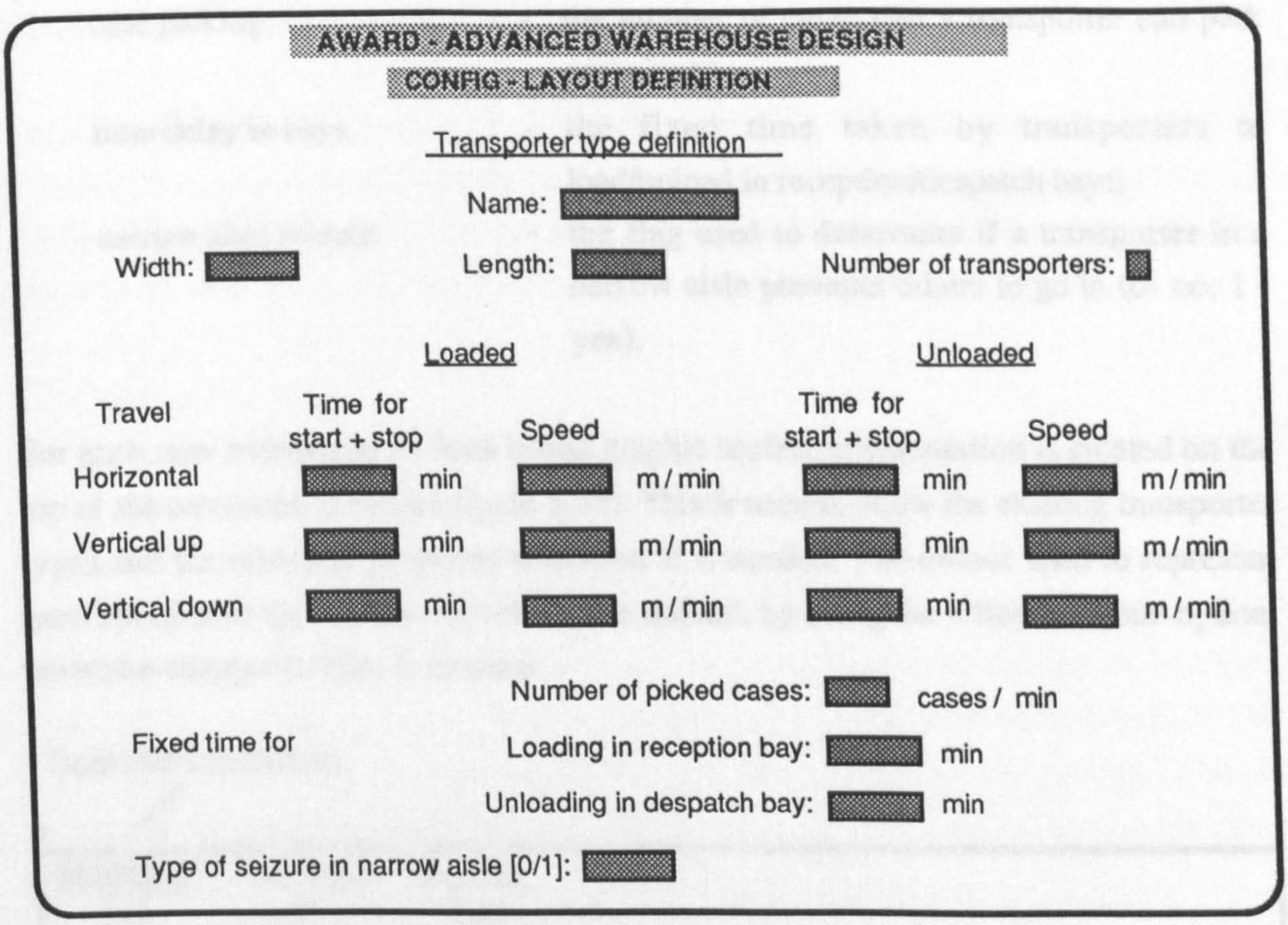


Figure 5.60 The transporter type definition data screen

These characteristics are entered in the data screen represented in figure 5.60. This screen was generated with the MSC utility (see 5.3.1.6.) and all the input/output control was made by the INSTA subroutine library (see 5.3.1.4.). For each transporter type it is necessary to define the following information:

name	- the transporter type name (10 characters);
dimensions	- the transporter width and length (metres) used to do a graphic scaled representation of the transporters;
nº of transporters	- the number of transporters of this type;
travel characteristics	- the total time (per travel) taken by the transporter to start and to stop its movement (min); the travel speed (m/min) along the horizontal, vertical up and vertical down directions when the transporter is loaded and unloaded;

case picking	-	the number of cases that a transporter can pick per minute;
time delay in bays	-	the fixed time taken by transporters to load/unload in reception/despatch bays;
narrow aisle seizure	-	the flag used to determine if a transporter in a narrow aisle prevents others to go in (0- no; 1 - yes).

For each new transporter an icon based graphic scaled representation is created on the top of the command area (see figure 5.61). This is used to show the existing transporter types and for selection purposes whenever it is needed. The colour used to represent each transporter type is the one selected as default, by using the 'Choose colour' option, when the transporter type is created.

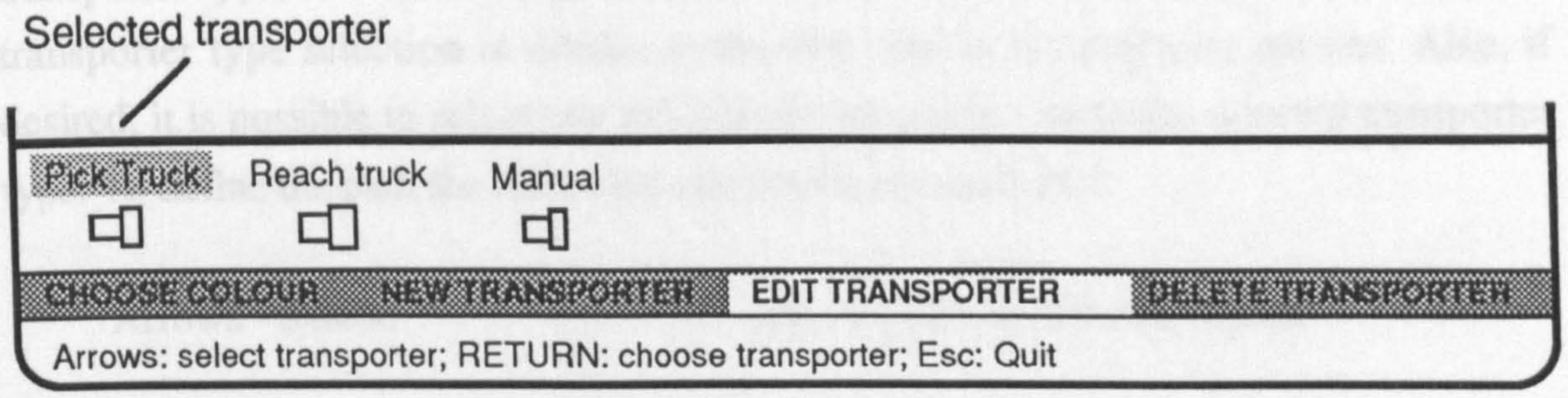


Figure 5.61 The edit transporter option showing the transporter type selection

. Edit transporter

With this option it is possible to edit the characteristics associated with any of the existing transporter types. After selecting this option it is necessary to choose the transporter type to edit. This is done by using the arrow keys to highlight the desired transporter type and then by pressing the 'Return' key to make the selection (see figure 5.61). Afterwards, a data screen, similar to the one of figure 5.60, is displayed with the transporter type data which can then be modified. To modify the transporter type colour it is just necessary to select the desired colour before entering the 'Edit transporter' option.

. Delete transporter

This option allows the deletion of any of the existing transporter types. The selection process of the transporter type to be deleted is similar to the one described in the previous option. After selecting the transporter type it is asked for the deletion to be confirmed and if that is the case the transporter type will be deleted.

. Define path

This option enables the definition of the transporter path. The transporter path is a set of aisles which the transporter can use to move itself within the warehouse. Through the use of the 'Define path' option, aisles can be added or taken from each transporter path. The same path can be associated with all the transporters, belonging to the same transporter type, or a different path can be defined for each individual transporter. The transporter type selection is similar to the one used in the previous options. Also, if desired, it is possible to select one individual transporter within the selected transporter type. To define the path the following commands are available:

Arrows - Select:	step through the different aisles;
RETURN - Add:	add the selected aisle to the path;
Rub out (delete) - Take:	take the selected aisle from the path;
A - All:	add all the aisles to the path;
N - None:	take all the aisles from the path;
Q - Quit:	quit this option and save the defined path;
Esc - Abort:	escape from this option without saving the defined path.

. Define park

This option is used to define the park position for each transporter. The transporter will go to the park position whenever it is idle and stays there until it is scheduled to another

job. The potential park positions are the nodes, from the transporter path network, that are only on a unique aisle. Looking at figure 5.50, it is easy to conclude that those nodes correspond to the aisle ends which are not intersected by any other aisle. After entering this option it is necessary to select one individual transporter. The selection process is similar to the one used in the 'define path' option but without the optional choice of making the definition to the whole transporters in the transporter type. To define the park position the following commands are available:

Arrows - Change park:	step through the available park positions;
RETURN - Set park and quit:	define the current park position as the selected transporter park position and quit this option;
Esc - Abort:	escape from this option without making any changes.

The potential park positions become unavailable the moment they are allocated to a particular transporter.

5.3.2.2. Working parameters definition

The working parameters data, as explained in 5.3.2., is related more to the warehouse operation rather than to the layout parameters as in the physical parameters data. In this section is described the role of the working parameters and the way they are defined. The working parameters are grouped under the following headers: racking zones - which are groups of racking cells where the products can be stored preferentially; reception bays - which are locations in the reception area where the trucks can unload; despatch bays - which are locations in the despatch area where the trucks can load; transporters jobs - which are the different jobs that a transporter can do within the warehouse; shifts - which are the periods of time during the working day to where the different resources can be allocated.

5.3.2.2.1. Racking zones definition

The racking zones are essential to the organisation of the different products within the warehouse. The products are grouped together in what is called product groups. Each product group can have up to three preferential zones with a decreasing level of priority. The priority level is used to decide the order in which the zones are searched to find free locations to store the product pallets. The racking zone definition is organized in two levels: in the first, the zones can be created, deleted or displayed; in the second, an existing zone can be selected and groups of cells can be created, edited or deleted within the zone. A group of cells is formed by a number of contiguous cells belonging to the same racking block. A racking zone can only be formed by a group of cells belonging to racking blocks with the same racking module. Figure 5.62 lists the available options to manipulate the racking zones.

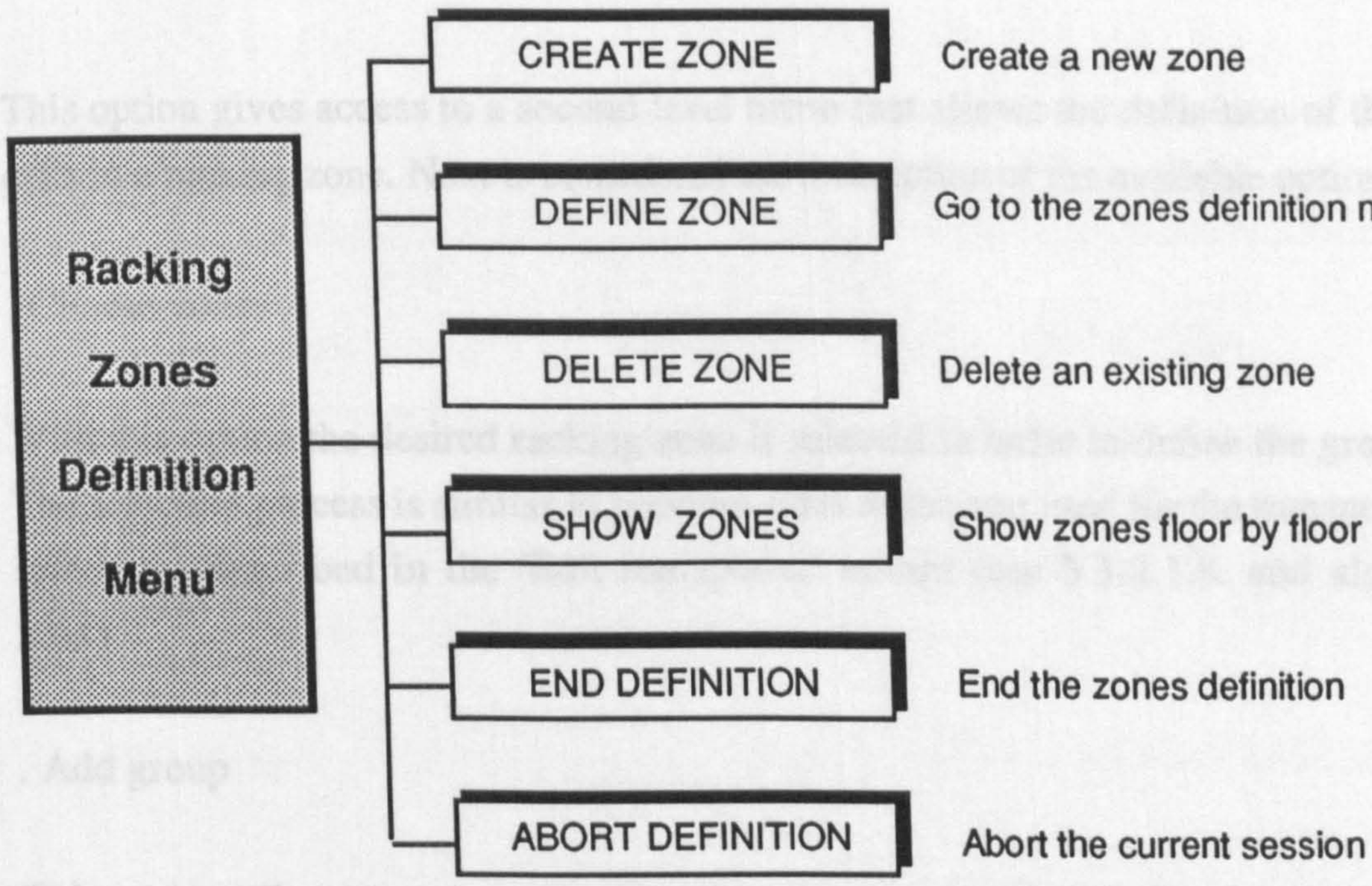


Figure 5.62 Racking zones definition menu

. Create zone

This option allows the creation of a new racking zone. After entering this option the following information must be defined:

zone colour - the colour used to represent the information related to each zone (e.g. name, cells etc.); the colour selection process is

the same as the one used in the 'Choose colour' option described in 5.3.2.1.1. (see also figure 5.16);

zone name - the name (10 characters) that is used whenever a zone must be adressed;

racking module - the racking module that specifies which racking blocks can be used to add groups of cells to the racking zone; the racking module selection process is the same as the one described in the 'New module' option (see 5.3.2.1.4.).

After creating a new zone its name is added to the list of zone names at the top of the command area (see figure 5.63). This list is used to display the existing zones and is used also, as part of the zone selection process.

. Define zones

This option gives access to a second level menu that allows the definition of the group cells in a racking zone. Next is considered the description of the available options.

. Choose zones

With this option the desired racking zone is selected in order to define the group cells. The selection process is similar to previous ones as the one used for the transporter type selection, described in the 'Edit transporter' option (see 5.3.2.1.8. and also figure 5.61).

. Add group

This option allows a group of cells to be added to the current selected racking zone. After entering this option only the racking blocks which have the same racking module as the one associated with the selected zone are displayed (see figure 5.63). To add cells to the racking zone the following commands are available:

RETURN - Set: set the current selected cell as the first cell of the group to be defined; add the defined group of cells to the racking zone and quit the option;

Arrows - Move: step through the cells within the racking block;

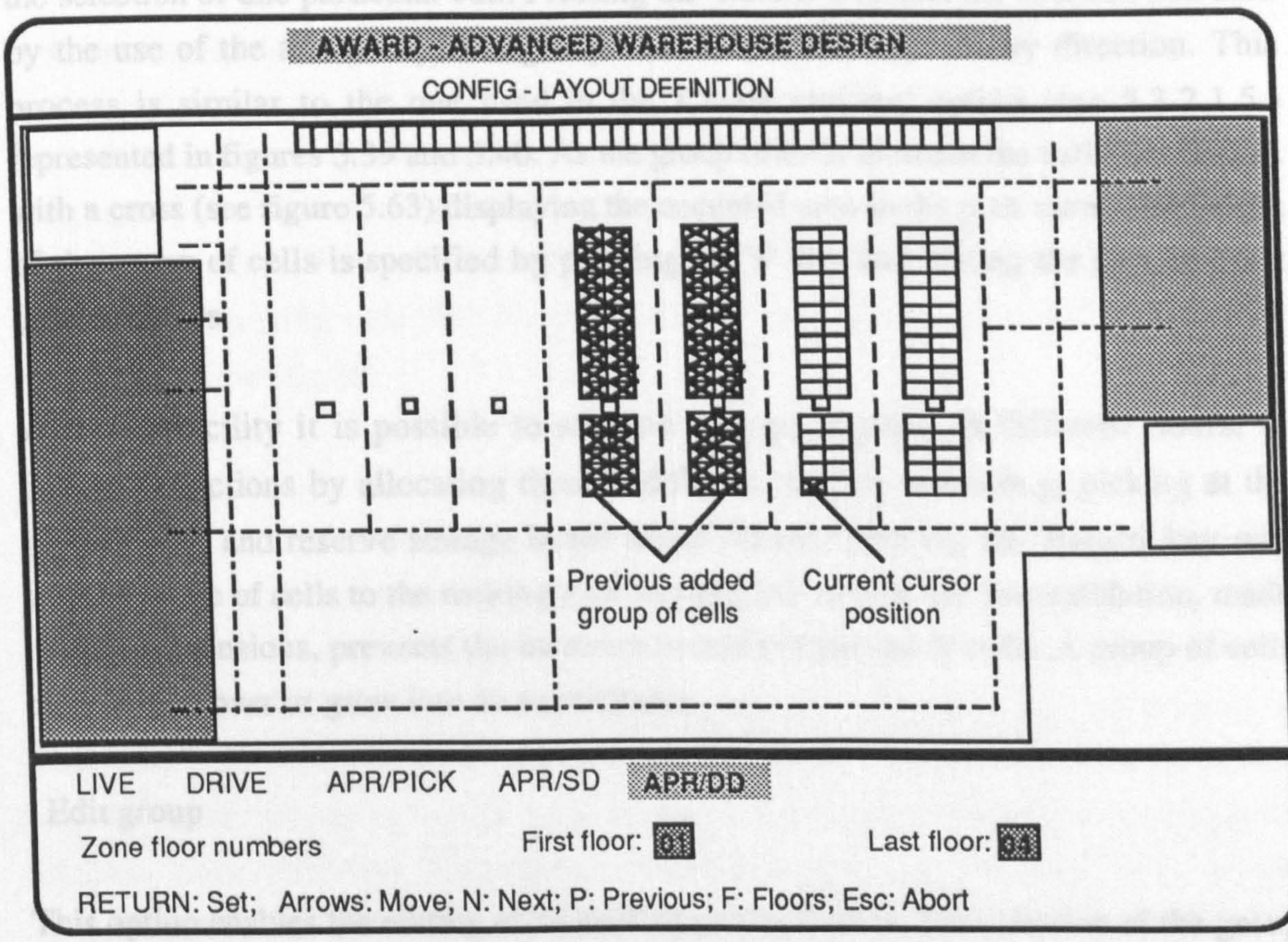


Figure 5.63 The warehouse layout showing the racking zones definition stage

N - Next: move the cursor to the next racking block;

P - Previous: move the cursor to the previous racking block;

F - Floors: enable editing of the first and last floor data fields;

Esc - Abort: escape from this option without adding the group of cells to the racking zone.

The 'N' and 'P' keys are used to position the cursor in the desired racking block. The cursor is represented by a cross that marks the cell (see figure 5.63). The arrow keys, joydisk or joystick are used to move the cursor within the racking block. This enables the selection of one particular cell. Pressing the 'Return' key sets the first cell and then by the use of the arrow keys the group of cells can increase in any direction. This process is similar to the one used in the 'Create racking' option (see 5.3.2.1.5.) represented in figures 5.39 and 5.40. As the group of cells increase the cells are marked with a cross (see figure 5.63) displaying the occupied area in the plan view. The height of the group of cells is specified by pressing the 'F' key and editing the first and last floor numbers.

With this facility it is possible to schedule groups of cells, in different floors, to different functions by allocating them to different racking zones (e.g. picking at the ground level and reserve storage in the above floors). Pressing the 'Return' key will add the group of cells to the racking zone and quit the option. On line validation, made in three dimensions, prevents the incorrect creation of groups of cells. A group of cells can not start over or grow into an existing one.

. Edit group

This option enables the editing of an existing group of cells. The selection of the group to be edited is made by using the 'N' and 'P' keys to step to the next and previous groups and the 'Return' key to make the selection. The selected group is highlighted by displaying the cells in the group marked with a cross (see figure 5.63). After selecting the group to be edited the procedure is similar to the one used in the 'Add group' option.

. Delete group

This option allows a group of cells to be deleted from a racking zone. The selection of the group to be deleted is similar to the one used in the previous option. After the selection it is asked for confirmation and the group is deleted if that is the case.

. Previous menu

Selecting this option returns the control to the first level menu.

. **Delete zone**

This option is used to delete an existing racking zone.

. **Show zones**

This option is used to quickly show the cells allocated to the different racking zones. As the groups of cells are three dimensional blocks a layer representation is used to display them in plan view. For each floor the cells are marked with a cross using the same colour of the racking zone they belong to. The 'N' and 'P' keys are used to select the next and previous floors.

5.3.2.2.2. Reception and despatch bays definition

The reception bays are the locations in the reception area where the trucks can unload pallets. The despatch bays are the locations in the despatch area where the trucks can load pallets. Any node from the transporter path network that belongs to just one aisle and are inside the reception or despatch areas can be defined as a corresponding reception or despatch bay. Although the reception and despatch areas definition is made in separate options only one description is given because of their similarity. The two main options for the bay definition are:

. **Create bay**

When the transporter path network data is generated, also identified are the nodes that can be defined as bays. In the 'Create bay' option the 'N' and 'P' keys are used to step through the available nodes and the 'Return' key is used to set the currently selected node as a bay. If a node is already allocated as a transporter park position or defined as a bay it is no longer available. A small square marker is used to highlight the already defined bays and the currently selected node.

. **Delete bay**

This option allows the deletion of an existing bay. Using a similar procedure to the one described in the previous option it is possible to step through the existing bays. Pressing the 'Return' key selects the desired bay which is deleted after confirmation.

5.3.2.2.3. Transporter jobs definition

This option assigns to each transporter type or individual transporter the job types it is able to do and their priority. The job type priority is used to decide the order in which the job types are scanned when an attempt is made to schedule a new job to a transporter. The following job types are considered:

- unload from reception bay to reserve storage;
- load from reserve storage to despatch bay;
- replenish picking zone;
- case picking to despatch bay.

The job types assignment can be made to the whole transporters in a transporter type or to each particular transporter. This selection is made using the same procedure as the one used in the 'Define path' option of the transporter definition (see 5.3.2.1.8.). For each transporter type or individual transporter it is necessary to specify: the number of different job types it is able to do; and the job types index. The job types index should be entered from high to low priority.

5.3.2.2.4. Shifts definition

The shifts are used to establish the availability of the warehouse resources during different periods of a working day. The shifts are defined by specifying the start and the finish time (HH:MM). The warehouse resources that can be made available/unavailable on the different shifts are: the transporters; the reception bays; and the despatch bays. The availability of the transporters can be specified by transporter type or by each individual transporter. The transporter selection is made as in the transporter jobs definition (see 5.3.2.2.3.). The reception bays availability can be specified as a whole or by each individual bay. The same applies to the despatch bays availability.

5.3.2.3. Product group definition

The product data is one of the most important pieces of information in a warehouse system. Sometimes it is not easy to gather all the information about each particular product, or the existence of products with similar characteristics does not justify considering each one individually. To facilitate the task of defining the product data it is possible to join similar products in what is called a product group. All the necessary information about each product group must be defined in a data file. The data is entered into the file using a normal text editor or the PG program. PG is an interactive program that allows the creation or modification of a product group data file through the use of data screens. For each product group the following information should be entered:

product group name	- the product group name;
nº of products	- the number of products in the group;
average nº of cases/pallet	- the value used as the mean in a normal distribution to define the nº of cases/pallet for each product;
SD nº of cases/pallet	- the value used as the standard deviation in a normal distribution to define the nº of cases/pallet for each product;
max. nº of reserve pallets	- the maximum number of product pallets that can be stored at the same time;
min. nº of reserve pallets	- the number of product pallets at which an inorder is automatically generated for new pallets to arrive;
first storage zone	- the name of the preferred racking zone reserve storage (high priority);
second storage zone	- the name of the second racking zone reserve storage (medium priority);
third storage zone	- the name of the third racking zone reserve storage (low priority);
picking zone	- the name of the picking racking zone;
pallet ROL (%)	- the percentage value that multiplied by the product nº of cases/pallet is equal to the nº of cases in the product picking location under which a replenish is automatically generated;
colour index	- the colour used whenever it is necessary to represent product information;

EOQ for pallets - the number of pallets (Economic Order Quantity) that is ordered when the nº of product pallets goes under the minimum number of reserve pallets;

time delay - the period of time between the generation of a inorder and its arrival.

5.3.2.4. Outorders definition

The outorders definition consists of the creation of a data file containing the characteristics of each order. The outorders data file can be created using real data or statistically generated data. The real data can be obtained from current or historical data records or from forecast data. The statistical data is generated using the OUTORDER program. OUTORDER is an interactive program that generates an outorder file based on user defined statistical parameters and on the information from a user specified product data file. For each order the following information is defined in the outorders data file:

inter-arrival time - the time remaining to the order arrival;
order number - the order number for user control;
number of lines - the number of lines in the order;
product code - the product code;
number of pallets - the number of required product pallets;
number of cases - the number of required product cases;

5.3.2.5. Generating the data for the simulation model definition

After defining all the required warehouse system data it is necessary to execute the option 'Create the initial running data file' before being able to run the simulation model. When executing this option it is assumed that the physical parameters data and the working parameters data are correctly defined. Each physical parameters configuration can have associated with it several sets of working parameters. Before executing this option one set of working parameters must be selected.

This option consists of three stages: the definition of the simulation initial parameters; the generation of the data for the simulation model definition; and the creation of the simulation model internal structure. The first two stages are part of the CONFIG

module and the last one forms the independent module IDUMP. The IDUMP module and the creation of the simulation model internal structure are described in section 5.3.3..

. Simulation initial parameters

At this stage the simulation start date and time are defined as well as the maximum period of time for running the simulation. This period of time is specified as a combination of weeks, days and hours. For the system to be able to generate the data for the simulation model definition a product and outorders file must also be selected.

. Generation of the data for the simulation model definition

During this stage the set of data consisting of the configuration, working and initial parameters is used to generate a database file with the required information for building the simulation model internal structure. The data is defined and written in different program modules under the following headers:

- Graphic data (ex: viewport and window coordinates, screen resolution, graphic scale)
- Transporters data (ex: no. of transporters and their characteristics, transporter paths and park positions, transporters jobs and priorities)
- Outorders data (ex: number of orders, maximum number of lines per order, maximum number of pallets per outorder, outorders arrival time)
- Despatch points (ex: no. of despatch points, despatch point coordinates, graphics data for trucks display)
- Reception points (ex: no. of reception points, reception points coordinates, graphic data for trucks display)
- Racking data (no. of rackings, racking coordinates, racking module type, module dimensions, no. of cells, module access logic)
- Product zones data (ex: no. of zones, maximum no. of cells per zone, zone name, zone colour, cells priority)
- Product group data (ex: no. of product groups, maximum no. of products per group, total no. of products, no. of cases / pallet, max. and min. no. of reserve pallets, product zones, pallet ROL, colour code, EOQ, lead time)
- Transporter paths adjacencies matrix
- Warehouse layout graphics data

After the creation of the database file the module IDUMP is automatically run to produce the simulation internal structure.

5.3.3. Model Definition Module

In this module is created the simulation internal structure for a specific configuration. The model structure is saved in a datafile which is called an "initial dump file". Different configurations can be set for the same warehouse layout (see figure 5.8) creating several "initial dump files" (see figure 5.4). As described in the simulation module, the warehouse configurations can be evaluated by running the simulation using the previously created 'initial dump files'.

The simulation internal structure is created within program IDUMP. The data in the database defined during the previous stage is used to create the simulation elements and initialise the simulation model. After this stage, the model is ready to be run as many times as required. The decision to build the model in a separate module is for efficiency reasons. The model is created just once so as not to overload the running of the simulation.

5.3.3.1. Simulation Elements

The simulation model structure is created based on the physical parameters, working parameters, products, outorders and initialization data. Most of this data is specified in a graphic form during the configuration stage. The use of graphics allows an easy and quick way of defining the warehouse data. Furthermore, it is possible to establish a relation between graphic objects and many of the simulation elements. There follows a description of the logic and the main components of the simulation model structure.

5.3.3.1.1. Simulation Executive

The SIMVIS simulation library subroutines (see 5.3.1.5) takes care of the logic of the simulation executive. When defining a simulation model, using SIMVIS, it is necessary to initialize the simulation overall parameters before being able to use the simulation subroutines. This initialisation is made by calling the subroutine INITZ0 with the

appropriate parameters. These parameters (see Appendix V.D) define the following information:

- maximum number of characters in the names of simulation elements (e.g. entities, queues, groups of entities and displays);
- number of initial, intermediate and final user interactions;
- maximum number of elements in the clock;
- screen coordinates and colours of the interaction box;
- maximum simulation time;
- number of user events;
- simulation execution delay.

Program IDUMP after reading the main parameters from the configuration database file calls the subroutine INITIZ0 to initialise the simulation model. A maximum of six characters are set for the names of simulation elements. The number of user interactions are set to one initial and eight intermediate. These user interactions are defined in the simulation module and are described later in this chapter. The maximum period of time for running the simulation is established according to the values in the database file. The maximum number of elements in the clock is set to one hundred, the number of user events is set to forty one and the simulation execution delay is disabled. The interaction box is set at the bottom right corner of the screen display.

Subroutine INITZ0 also defines two entities: the end of simulation entity ENDSIM and the interaction event entity INTEVE. ENDSIM is used to schedule the END OF SIMULATION event and it has one numeric attribute which is the maximum period of time for running the simulation. INTEVE has one numeric attribute which is the interaction event flag. If this flag is on, the simulation is interrupted after the execution of each event, and the user can control what happens next through the commands of the interaction box. This is very useful during the debugging phase.

For displaying the current date and time on the screen during the running of the simulation a set of calendar entities are created. These entities which are defined in the CRICAL subroutine are the following ones:

CAL - Calendar entity;
MES - Month entity;
DIA - Day entity;
HOR - Hour entity;
MIN - Minute entity;

The CAL entity attributes are: the number of days for each month of the year, the current year, month, day, hour and minute and pointers to the other calendar entities (numeric attributes) and strings with the initial and current dates and the month names (alphanumeric attributes). The MES, DIA, HOR and MIN entities all have just one numeric attribute which is a pointer to the CAL entity. Two events are scheduled one for handling the next minute change and the other for handling the next hour change.

5.3.3.1.2. Global System data

The SYSTEM entity, created in program IDUMP, is used to store global information and pointers to data which is frequently used. The data stored in entity SYSTEM includes the following items:

Movie Flag	-	animation control flag (ON/OFF); controls if the screen display is or is not updated during the simulation;
Transporter display increment	-	time interval for updating the position of the transporters on the screen display;
Current dump number	-	number of the dump which was used to start running the simulation;
Number of product groups	-	total number of product groups;
Maximum no. of products/group	-	maximum number of products in the product groups;
Total number of products	-	total number of products in all the product groups;
Application name	-	name of the current configuration file;
Product file name	-	name of the file where the product data is stored;
Outorders file name	-	name of the file where the outorders data is stored;
Graphics file name	-	name of the file where a copy of the simulation initial graphic screen display is saved;

In Appendix V.F the other attributes of the SYSTEM entity are described.

5.3.3.1.3. Graphics Data

The Graphical Information entity (GRAINF), created in the subroutine with the same name, stores the required data to display graphic information. The GRAINF entity attributes, listed in Appendix V.F, store the parameters of the following items:

Viewport 1 coordinates	- the viewport graphic coordinates of the simulation display;
Viewport 2 coordinates	- the viewport graphic coordinates of the simulation interaction commands display;
Window 1 coordinates	- the window graphic coordinates of the simulation display;
Window 2 coordinates	- the window graphic coordinates of the simulation interaction commands display;
Graphics scale	- the scale parameters used to transform world coordinates to virtual coordinates;
Screen resolution and ratio	- the pixel resolution of the graphic display and the ratio between the screen width and height.

5.3.3.1.4. Transporters

The simulation elements related to the transporters are created in subroutine CRETRA. The characteristics of the different type of transporters, defined in the configuration module, are kept as attributes (see Appendix V.F) of corresponding transporter type entities with the generic name TRTY11, where 11 stands for a two digit sequential number for the transporter type (i.e. TRTY01, TRTY02 etc.). These entities are added to the transporter type processing queue (TRTYQU) and remain there during the simulation run. A transporter type waiting queue, with generic name QUTY11, is created for each transporter type and used to keep track of each transporter of that type.

A transporter entity is created for each individual transporter within each transporter type. The generic name of the transporter entity is TR1122 where 11 is a two digit transporter type number and 22 stands for a sequential two digit number of the transporter within that type (e.g. TR0204 is the entity name of the 4th transporter within transporter type no. 2). The transporter entities are added to the queue QUTY11 of the

corresponding transporter type (e.g. transporter entity TR0204 is added to the queue QUTY02).

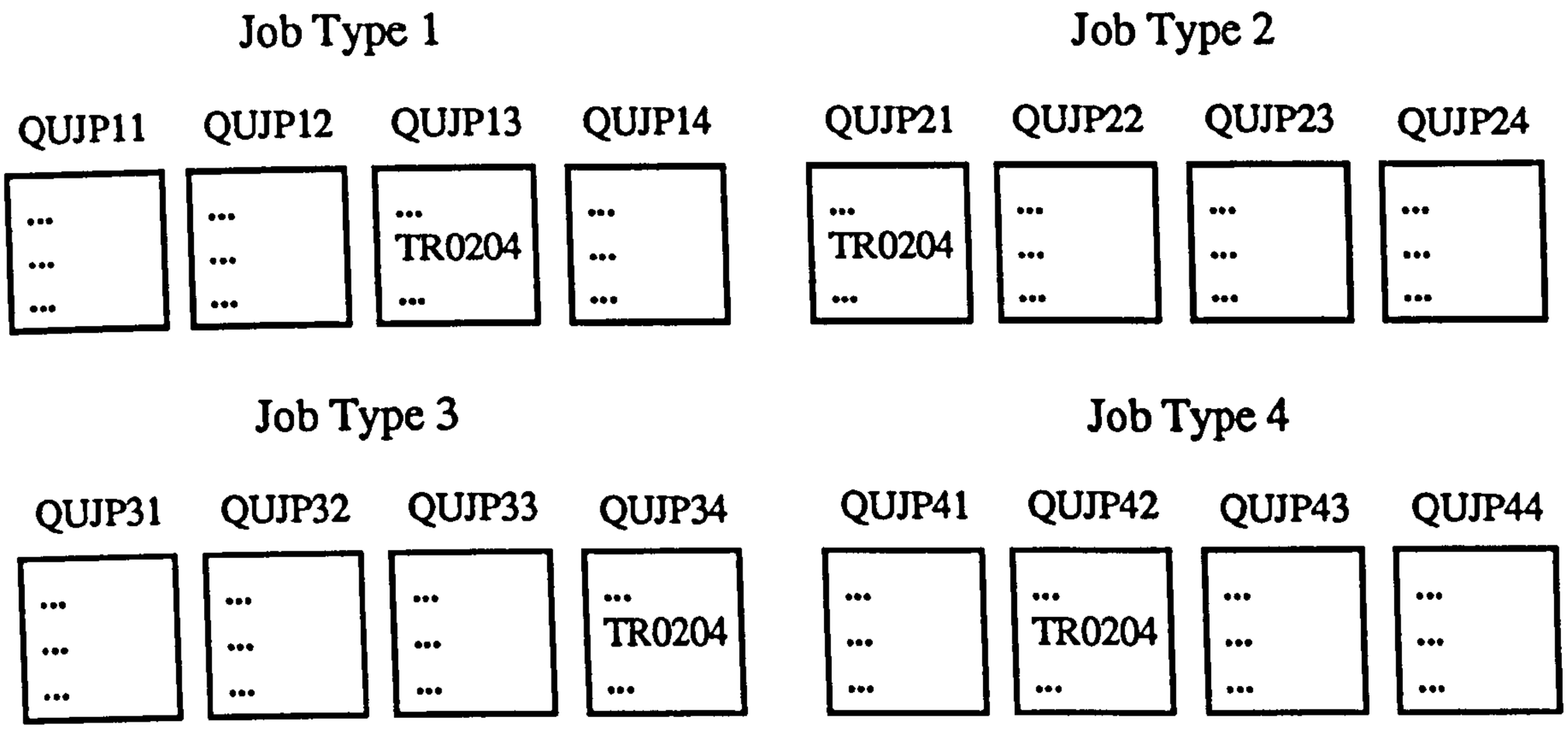
New simulation elements are created, in subroutine CRETRA, to implement the transporters ability to handle several type of jobs with different level of priorities. A job type entity (JOBTY1) is created for each different type of job. These entities are added to the job type queue (JOBTQU) and remain there during the simulation run. For the software version described here four job type entities (JOBTY1,...,JOBTY4) are created corresponding to the following jobs:

- unload from reception bay to reserve storage;
- load from reserve storage to despatch bay;
- replenish picking zone;
- case picking to despatch bay.

For each job type four job priorities queues (a number equal to the number of job types) are created (see Fig. 5.64). The queues are named QUJP12 where the first digit (1) is the job type number and the second (2) is the queue order (priority) number. For a certain job type: the transporter entities corresponding to transporters which have that job type as first priority will be added to the first queue; the transporter entities corresponding to transporters which have that job type as second priority will be added to the second queue and so on.

During the simulation run when a job, of a certain type, is generated these queues are used to decide which one from the available transporters will be scheduled to take that job according to the defined priorities. For example, if a job of type 2 is generated the queues QUJP21 through QUJP24 will be sequentially inspected and the first transporter entity found (if any) will be picked to do the job. The addresses (pointers) of the job priority queues, for each job type, are stored in a vector. A pointer to this vector is saved as an attribute of the corresponding job type entity.

For each transporter entity a vector is created with the addresses (pointers) of the job priority queues corresponding to the job types the transporter can handle and in descending order of the job priorities. For example, if the transporter TR0204 can handle the four job types with the priorities described in Fig. 5.64, the vector elements point in sequence to the queues QUJP21, QUJP42, QUJP13 and QUJP34.



The transporter TR0204 has the following job priorities:
job type 2 (1º)
job type 4 (2º)
job type 1 (3º)
job type 3 (4º)

Figure 5.64 The job priorities queues for each job type

If the transporters are available at the shift at the start of the simulation run, the corresponding transporter entities are added to the job priority queues, otherwise they are added to the inactive transporters queue (QTRINA). A shift times vector is created for each transporter entity with the times when there are changes in the transporter availability. A virtual transporter entity is created (FR1122), for each transporter entity, to store the shift data and to be used in scheduling the shift change events. If necessary, the events handling the transporter start or stop working at the next shift change, from the beginning of the simulation, are schedule with the FR1122 entity.

Another queue is created, for each transporter entity, called the jobs waiting queue (QJ1122). This queue is used to store all the jobs that are already scheduled (normally just one) to the transporter. A predetermined number of job entities (TJ1111) are created and added to the jobs neutral queue (QNTJOC). These entities are used to store the characteristics of each job generated during the simulation run. Two vectors are created in association with each job entity: the transporter cell entity pointers; and the transporter case picking entities. These vectors are used by the transporter, during its travel, as case picking lists to determine which is the next racking cell to go to, and how many cases should be picked.

A predetermined number of transporter path entities (TP1111) are created and added to the paths neutral queue (QWTRAP). The path entities are used to described the line segments of a transporter path. During the simulation run, when a transporter needs to travel from one point to another, its path is divided into line segments whose characteristics (initial and final coordinates, direction cosines, length, etc.) are saved as attributes of the path entities. These entities, describing the transporter path, are then added to the transporter path queue (QP1122) and used later to move the transporter.

Three more queues are created in subroutine CRETRA to be used during the simulation run: the jobs processing queue (QPTRAJ), used to store the jobs being processed; the transporter movement queue (QTRAMO), which stores the transporters that are in motion; and the transporters processing queue (QPTRAN) which stores the transporters that are processing a job.

5.3.3.1.5. Outorders and despatch bays

The simulation elements related to the outorders and despatch bays are created in subroutine CREOUT. An outorder inter-arrival times vector is created to store the outorder arrival times, which are read from the configuration database file. A predetermined number of outorder entities (ORD111) are created and added to the outorders neutral queue (QOUTOR). These entities are used to store the outorder characteristics (e.g. time of arrival, no. of pallets and cases, start and finish processing times, etc.).

The outorder control entity (ORDCON) is used to schedule the outorder arrival events and controls the outorders arrival sequence and the outorders waiting for a despatch bay. In subroutine CREOUT an outorder arrival event is scheduled, with the ORDCON entity, for the time of arrival of the first outorder.

A despatch bay entity (DIS111) is created for each warehousing despatch bay. The despatch bay entity attributes (see Appendix V.F) store some bay characteristics (e.g. position coordinates, index, branch number, etc.) and also pointers to related simulation elements. For each despatch bay entity two queues are created: the outorder pallets waiting queue (QPW111) where the outorder pallets wait for transport; and the outorder cases waiting queue (QCW111) where the outorders cases wait for transport.

A virtual despatch bay entity (FIS111) is also created for each despatch bay. The virtual despatch bay entity handles the changes in bay availability between shifts. The times when there are changes in the bay availability are stored in a shift times vector whose address (pointer) is saved as an attribute of the virtual despatch bay entity. One of the two events: bay enters into activity; or bay retires from activity; is scheduled with the virtual despatch bay entity depending on the bay availability at the start of the simulation.

Three queues are created in relation with the despatch bays: the despatch bays waiting queue (QNEDIS) which is used to keep the despatch bay entities when they are idle; the despatch bays inactive queue (QINDIS) which is used to keep the despatch bay entities when they are not available in one shift; and the despatch bays processing queue (QPRDIS) which is used to keep the despatch bay entities when busy (i.e. a truck is unloading at the despatch bay). If the despatch bays are available in the shift at the start of the simulation run, the corresponding despatch bay entities are added to the despatch bays waiting queue (QNEDIS), otherwise they are added to the inactive despatch bays queue (QINDIS).

A predetermined number of pallets entities (PAL111) and case entities (CAS111) are created and added to the pallets neutral queue (QUENPA) and cases neutral queue (QUENCA). These simulation entities are used when pallets or cases need to be moved from one place to another (e.g. from the despatch bays to the racking).

5.3.3.1.6. Inorders and reception bays

The simulation elements related with the inorders and reception bays are created in subroutine CREREC and are almost all equivalent to those described in 5.3.3.1.5. for the outorders and despatch bays. For this reason there is not proceeded a detailed description of the inorders and reception bays simulation elements. The logic is the same as the one described in 5.3.3.1.5. and in Appendix V.F the name and the characteristics of the new simulation elements are presented. Nevertheless, there are some differences that must be pointed out.

The inorders are always formed by full pallets and not by individual cases. So, for each reception bay entity (REP111) just the inorder pallets waiting queue (QPI111) is

created; there is no equivalent to the outorder cases waiting queue (QCW111). As it is obvious, subroutine CREREC does not create the pallets entities (PAL111) and the pallets neutral queue (QUENPA) as they already exist.

5.3.3.1.7. Racking

The simulation elements related to the pallet racking are created in subroutine CRERAC. A racking entity (RAC111) is created for each contiguous group of cells with the same characteristics. These entities are added to the rackings neutral queue (QUERAC). The racking entity attributes (see Appendix V.F) store the racking physical characteristics (e.g. physical coordinates, direction cosines, no. of cells, cell dimensions, no. of floors high, etc.) and pointers to related simulation elements.

Several vectors are created in subroutine CRERAC to support the racking logic. The racking access input / output cell indices vectors and the racking access input / output branch indices vectors relates each cell to the aisle branch which give access to it for input / output. This information becomes necessary, during the simulation run, to define the transporter path from its current position to the position where it can access the racking cell.

If the racking is a powered mobile type of racking (PMR) the PMR waiting transporter vector is created to act as a queue for the transporters waiting to access the racking. As it is explained in 5.3.2.1.4. the way PMR type of racking works, with moving parts (see Fig. 5.34), only allows one transporter to access the racking at a certain time. If other transporters want to access the same racking they must wait in a queue for their turn to arrive.

5.3.3.1.8. Racking Zones and Cells

The simulation elements related to the racking zones and racking cells are created in subroutine CREZOC. The zones entities (ZON111) are created and added to the zones queue (QUEZON). Each zone entity represents a group of racking cells with identical stock layout characteristics. The zone entity keeps as attributes, during the simulation run, the current number of free and occupied cells in the zone.

A cell entity (C11111) is created for each cell within a zone. The cell entities are added to the zone free cells queue (QZO111) whose pointer is saved as a zone entity attribute. The zone free cells queue has, during the simulation run, the zone cell entities which are empty. The cells are sorted in the queue by a sort key number which is determined by the way the racking and the zone are created during the configuration stage. So, the cell entity at the top of the queue is the best choice for all the free cells according to the defined priority criteria.

The cell entity attributes (see Appendix V.F) store the characteristics (e.g. maximum and current number of pallets in the cell, no. of cases, sort key number, etc.) and pointers to related simulation elements (e.g. racking entity, product entity, zone entity).

5.3.3.1.9. Products

The simulation elements related to the products are created in subroutine CREPRO. The product group entities (PG1111) are created and added to the product groups neutral queue (QPRGRO). The product group entity stores as attributes (see Appendix V.F) the characteristics of the product group (e.g. no. of products in the group, maximum and minimum no. of reserve pallets, etc.) and pointers to related simulation elements (e.g. reserve storage zone entities, picking storage zone entity, etc.).

The inorder products waiting queue (QWAPRO) is created to store the inorders generated by the products when the number of pallets stored go down to the minimum level. The products waiting for replenishment queue (QWAPRE) are created to store the replenishment orders issued when the number of product cases, at the picking cells, goes down to the minimum level.

A product entity (PR1122) is created for each product within the product group. The first digits in the product entity name represent the group and the last digits the product sequential number. The number of digits (11) and (22) depends on the maximum number of product groups and on the maximum number of products per group. The product entities are added to the products neutral queue (PQ1111) where (1111) is the number of the corresponding product group.

The product entity has as attributes (see Appendix V.F) the characteristics of the product (e.g. no. of cases per pallet, no. of pallets), pointers to related simulation elements (e.g.

product group entity, picking cell entity, etc.) and control and statistics data (e.g. replenishment flag, no. of cases out of stock, etc.). A cell entity pointers vector is created for each product entity. This vector is used, during the simulation run, to store the pointers to the racking cells with the product pallets. A pointer to this vector is saved as an attribute of the product entity.

In subroutine DEPICE it is attributed to each product a picking cell. This operation is done for each product group by taking a cell from the product group picking zone free cells queue for each product in the group.

5.3.3.1.10. Transporters Movement

The simulation elements related with the transporters movement are created in subroutine CRETMO. The transporter movement entity (TRAMOV) is created to manage the transporters path network data. The TRAMOV entity has as attributes (see Appendix V.F) the number of nodes and branches (see Fig. 5.49 e)) in the path network and pointers to the vectors that store the path network data.

The node x coordinates vector and the node y coordinates vector store the (x,y) coordinates of the path network nodes. The branch left nodes vector and the branch right nodes vector store the two node numbers of each branch in the path network. The branch lengths vector stores the length (m) of each branch. The adjacent node indices vector stores the numbers of the adjacent nodes for each node in the network. The adjacent node branch indices vector stores the number of adjacent branches for each node in the network. The adjacent node pointers vector stores the pointers to the position, in the two previous vectors, where the data corresponding to each node in the path network is stored. The narrow aisle entity pointer vector stores the pointers to the narrow aisle entities.

The narrow aisle entities (NA1111) are created in subroutine REAMOV and are used to implement the logic of the warehouse aisles where just one transporter can get in at the same time. The narrow aisle entity attributes are the occupation flag, which signals if a transporter is in the aisle, the pointer to the first transporter entity waiting to get into the aisle, and the number of transporters waiting to get into the aisle.

5.3.3.1.11. Logical Displays

Four logical displays are created in subroutine CRELOD. Logical display no. 1 corresponding to the warehouse layout; logical display no. 2 corresponding to the pallet display in the racking; logical display no. 3 corresponding to the transporters; and logical display no. 4 corresponding to the inorder and outorder trucks. The logical display is a simulation element which stores a set of information, usually with logical links, and can be set on or off to display or not display the information on the screen.

5.3.4. Simulation Module

The simulation module allows the running of any model created during the configuration stage. The simulation can start (see Fig. 5.4) from an "initial dump", created by the program IDUMP, or from an "intermediate dump" created during a previous run of the simulation model. The program MODEL implements the SIMVIS simulation executive (see Fig. 5.2) which controls the execution of the simulation events and makes available the system and user interactions.

5.3.4.1. Simulation Executive

The program MODEL starts by an initialisation phase. It resets and does the necessary settings to the graphics terminal. Then, in subroutine INISIM, it prompts the user for the application name and reads and displays the application available "dumps". Afterwards the user can choose any of the "dumps" to start the simulation from. Following the subroutine INISIM restores the simulation parameters, displays the graphics information (e.g. warehouse layout, pallets, transporters, etc.) and returns to program MODEL.

Subroutine INTERA, which controls the interactions execution, is then called. When the user chooses to start the simulation, giving the RUN command, the execution returns to program MODEL which enters in the simulation executive phase. It handles the events execution and if the user presses any key it calls subroutine INTERA and makes the interactions available again to the user. This cycle is repeated until the simulation reaches the end or the user stop it using the interaction command KILL. If

necessary during this phase the transporter positions are updated on the screen graphics display.

5.3.4.2. Interactions

Interactions are a set of commands that become available when the simulation run is interrupted by pressing any key on the keyboard. The available interaction commands are listed, during the simulation run, in a menu at the bottom of the screen (see fig. 5.65). The two bottom lines show the system interactions which are the commands that control the simulation and are common to other SIMVIS models. The two first lines are reserved for the user interactions which are commands directly related with the model being run.

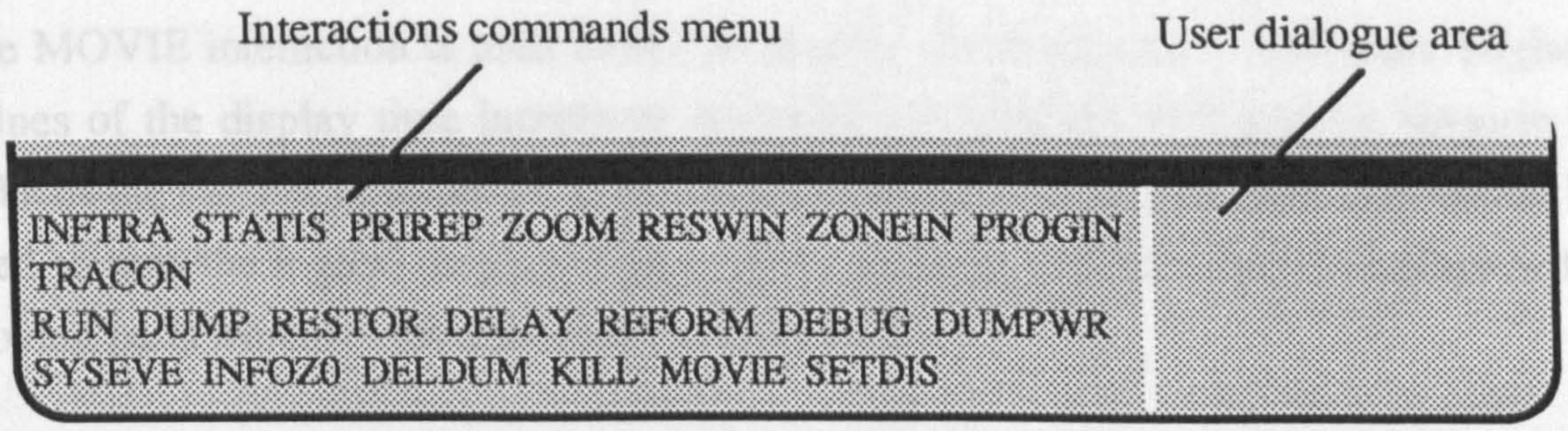


Figure 5.65 The interaction box with the system and intermediate user interactions

The user interactions are divided into three groups: the initial user interactions available only at the beginning of the simulation; the intermediate user interactions available during the running of the simulation; and the final user interactions available only at the end of the simulation. The interactions commands input is made in subroutine INTERA which allows editing facilities and after validation calls the appropriate subroutine to handle the given command.

5.3.4.2.1. System Interactions

Subroutine INTSYS implements the following system interactions:

RUN - start or restart the simulation run;

DUMP - copy the internal memory simulation structure to a disk file called intermediate dump file;

RESTOR - restore the internal memory simulation structure from a previous saved initial or intermediate dump file;

DELAY - delays the model execution for a user defined value;

REFORM - redisplays the information of all logical displays which are ON;

DEBUG - displays on the screen tracing information;

DUMPWR - writes to a file the simulation internal structure data;

SYSEVE - interrupts the simulation at the beginning of each event;

INFOZ0 - gives access to a set of menus to display simulation data;

DELDUM - deletes existing "dump files";

KILL - stops the model run and ends the simulation session;

MOVIE - defines the transporter display time increment;

SETDIS - turns On or OFF any of the logical displays.

The MOVIE interaction is used to set the transporter display time increment. Higher values of the display time increment increases the graphics performance because it reduces the number of times the graphics screen is updated. The SETDIS interaction is used to turn the logical displays ON or OFF. Turning OFF the logical displays with moving parts increases the simulation model speed.

5.3.4.2.2. Initial User Interactions

Subroutine INTINI controls the calling of the subroutine which implements the selected user initial interaction command. In this version just one user initial interaction is available. This interaction, called FILL, is implemented in subroutine INI001. The interaction FILL allows the user to do an initial fill of the warehouse starting the simulation from a more realistic situation.

The fill algorithm starts by filling with pallets the product picking cells. Then it allocates cells in the reserve storage and fills them with pallets until each product minimum reserve storage value is reached. The final stage of the fill algorithm allocates cells in the reserve storage and fills them with pallets for a value which is calculated as a percentage, given by the user, times the difference between the maximum number and minimum number of reserve pallets for each product.

5.3.4.2.3. Intermediate User Interactions

Subroutine INTINT controls the calling of the subroutine which implements the selected intermediate user interaction command. The intermediate user interactions are implemented in subroutines INT001, INT002, ... corresponding to each one of the following commands:

INFTRA - displays graphic bar charts of transporter performance statistics;
STATIS - displays a screen of current warehouse performance statistics;
PRIREP - prints to a file the current warehouse performance statistics;
ZOOM - allows the interactive definition of the size and location of the zoom window;
RESWIN - restores the default window after zooming in;
ZONEIN - displays graphics bar charts with information about zones;
PROGIN - displays graphics bar charts with information about products;
TRACON - controls the transporters activity;

5.3.4.3. Events

As it is described in 5.2.1. AWARD simulation executive (see Fig. 5.2) is implemented using the next event technique (see 2.1.) and the event approach modelling (see 2.2.2.). With the next event technique the simulation model progress from one event to the next event in a time ordered fashion. With the event approach modelling the events handle the logic involved with the changes in the internal state of the simulation model. In program MODEL each event is implemented in a separate subroutine called EVEN11 where (11) is the event number. The execution of the scheduled events is controlled by the simulation executive and by subroutine USEEVE which handles the call of the subroutines corresponding to each event to be executed.

An event list is presented in Appendix V.F with the event number, a brief event description and the entity name associated with the event scheduling. These events can be grouped according to their related activities under the following headers:

simulation end - the last event to be executed;
date - minute, hour, day, month and year updating;
transporters control - transporter movement and job scheduling control;
outorders - outorder arrival, initialisation, processing and finish;

loading job	-	taking pallets from reserve storage to despatch bay;
picking job	-	picking cases and taking them to despatch bay;
replenishment job	-	taking pallets from reserve storage to picking cell;
unloading job	-	taking pallets from reception bay to reserve storage;
inorders	-	inorder arrival, initialisation and finish;
parking	-	transporter returning to park;
breakdown	-	transporter breakdown;
shifts	-	transporter and bays shift changes;
reports	-	generating and scheduling reports.

5.3.4.3.1. Simulation End

The "End Simulation" event (event no. 1) is executed at the end of the simulation run. This event is automatically scheduled by program IDUMP for the maximum period of time for running the simulation which is defined by the user as a simulation initial parameter (see 5.3.2.5.). The user can also end the simulation run at any time by using the system interaction KILL.

5.3.4.3.2. Date

The "Change Hour" event (event no. 2), "Change Day" event (event no. 3), "Change Month" event (event no. 4) and "Change Minute" event (event no. 5) take care of the time and date change and they also update the simulation clock on the screen display. These events are scheduled with the entities with the same name (see events list in Appendix V.F).

5.3.4.3.3. Transporter Control

This group of events includes event no. 7 "Free transporter", event no. 8 "Start a transporter job and update the transporter statistics" and event no. 9 "Move a transporter one increment of simulation time". Event no. 7 determines what will be the next job to be scheduled to a transporter when it becomes idle. Event no. 8 updates the transporter statistics and schedules the next event depending on the transporter job to be executed. Event no. 9 is executed on every unit of simulation time while the transporter is

travelling it checks if the transporter can move to the next position and updates its position.

. Event no. 7 - Free transporter

This event is scheduled with the transporter entity for the time when the corresponding transporter becomes idle. The decision rules used in this algorithm have resulted from the opinions and discussions with many warehouse professionals during the development of this work.

The main objective of this event is to find, from the pending jobs, one that the transporter is able to do and has the highest priority. The job is assigned to the transporter using a job entity which is added to the transporter jobs queue. The job number and the pointer to the transporter are saved as attributes of the job entity. If a job is assigned to the transporter then event no. 8 "start a transporter job" is scheduled with the job entity.

If the transporter has the breakdown flag set then job no. 6 (return to park breakdown) is assigned to the transporter and the event no. 8 is scheduled. Otherwise a loop for each transporter job type, in descending order of priority, is executed until a job is assigned to the transporter, or every job type has been checked. The variable "iflag" is used to detect if a job is already assign to the transporter, bypassing the following instructions and abandoning the loop.

In the case of job type no. 1 (unload pallets from reception bays to reserve storage) the following rules are used to assign a job to the transporter. First, if the transporter is already assigned to a reception bay the subroutine SPAREC is called to try to allocate to the transporter a pallet waiting in that bay. Second, for each reception bay and if the bay does not have a transporter assigned to it, call subroutine SPAREC to try to allocate a pallet to the transporter. Third, for each reception bay, and if the bay has a transporter assigned to it, call subroutine SPAREC to try to allocate a pallet to the transporter.

Subroutine SPAREC allocates to the transporter a pallet waiting in a reception bay. It starts by calling subroutine TRABAC to check if the transporter has access to the reception bay aisle. If it does, then for each pallet in the reception bays waiting queue, it calls subroutine CEBRAC to determine the cell access aisle. Afterwards, it calls subroutine TRABAC to check if the transporter has access to the cell aisle. If it does, it

takes the pallet from the waiting queue and put it in the pallets neutral queue; if the transporter was idle it takes it from the jobs priorities queue and puts it on the processing transporters queue; assigns the transporter to the reception bay if it was not already assigned; takes a job entity from the neutral queue, defines its attributes (job no. 1 and pointer to the transporter) and adds it to the transporter jobs queue; and finally schedules event no. 8 with the job entity.

In the case of job type no. 2 (load pallets from reserve storage to despatch bays) the rules used are equivalent to the rules used for job type no. 1, but subroutine DPATRD is called instead of subroutine SPAREC. Subroutine DPATRD is equivalent to the subroutine SPAREC but its objective is to allocate to the transporter a pallet in a despatch bay outorder pallets waiting queue.

In the case of job type no. 3 (replenishment pallets to picking cells) for each product in the replenishment queue the subroutine CHEFCE is called to check if there are pallets available on the reserve storage zones for the product. If a pallet is found then the subroutine DERTCE is called to allocate a pallet to the transporter. Subroutine DERTCE is equivalent to the SPAREC and DPATRD subroutines.

In the case of job type no. 4 (case picking) the rules used are equivalent to the rules used for job types no. 1 and no. 2, but subroutine DCATRD is called instead of subroutines SPAREC and DPATRD. Subroutine DCATRD is equivalent to the SPAREC, DPATRD and DERTCE subroutines.

If no job is assigned (iflag = 0) to the transporter then, if the transporter is idle, it is taken from the processing queue and added to the job priorities queues. Otherwise, if the transporter has just finished a job, then job no. 5 (return to park) is assigned to the transporter, and event no. 8 is scheduled.

. Event no. 8 - Start a transporter job and update the statistics

Event no. 8 updates the transporter statistics for the job just finished by the transporter and schedules the first event of the sequence of events implementing the job assigned to the transporter. It starts by taking the job entity from the transporter jobs waiting queue and add it to the jobs processing queue. Then it updates the transporter attribute corresponding to the total time doing jobs of the type of job just finished.

Afterwards, depending on the job assigned to the transporter, it schedules the appropriate event (the first one of the events sequenced in table 5.1) and defines the transporter attributes: occupation flag and initial operation time. Table 5.1 describes, for the different jobs, the occupation flag value, and the event sequence numbers which implement the job.

Job number	Job description	Occupation flag	Event sequence
1	Unload pallets from reception	1	32, 33, 34, 35, 36
2	Load pallets to despatch	2	14, 15, 16, 17
3	Replenishment picking cells	3	25, 26, 27, 28, 29
4	Case picking	4	20, 21, 22, 23, 24
5	Return to park (idle)	-2	18, 19
6	Return to park (breakdown)	0	38, 39

Table 5.1 Event sequence and occupation flag for the different transporter jobs

Job no. 6 is used for both situations where the transporter is returning to park because it has a breakdown or either it is retiring from activity (in the ending of a shift). The two situations are distinguished by the transporter activity flag attribute. If the transporter is active then it is a breakdown, otherwise the transporter is retiring from activity.

. Event no. 9 - Advance a transporter one increment of simulation time

Event no. 9, scheduled with the job entity, updates the position of the transporter, to which the job is assigned, and it checks if the transporter can move, on the next increment of the simulation time. It starts by checking if the transporter was free to move from the previous to the current unit of simulation time. If it was free to move, its position coordinates and the length along the current path are updated. If the transporter was not able to move then the transporter waiting time is updated accordingly.

Event no. 9 checks if the transporter can move to the next position by testing for collision with the other transporters in the processing queue. If a collision is detected, then with a recursive algorithm it checks if there is any circular reference to the same transporter. If there is, it means that the transporter is not able to move because of

another transporter or transporters that are blocked by itself. In this case, the transporter is allowed to move to solve the dead end. Event no. 9 also handles the narrow aisle situation, where a transporter must wait at the entrance of a narrow aisle if another transporter, which seizes the narrow aisle, is already there.

When a job is assigned to a transporter, the travel path from the initial to the final positions is computed. The travel path is divided in individual paths which does not cross any other aisle. The characteristics of each individual path are saved as attributes of a path entity which is added to the transporter paths queue. When the transporter reaches the end of an individual path, subroutine COPTRA is called to define the transporter attributes corresponding to the new path characteristics. The initial and final positions of the new path must be calculated because, depending on the aisle type and transporter direction, the transporter moves along the middle of the aisle or is shifted to the left or to shifted the right. Subroutine COPTRA also updates on the screen the transporter position (rotation) when from one individual path to another it changes direction.

When the end of the last individual path is reached by the transporter, event no. 9 takes the transporter from the movement queue, updates its position, and schedules the event whose number is a transporter entity attribute (23). Otherwise event no. 9 schedules it self for the next unit of simulation time.

5.3.4.3.4. Outorders

This group of events includes event no. 10 "Arrival of an outorder", event no. 11 "Outorder initialisation", event no. 12 "Start processing an outorder" and event no. 13 "Finish processing an outorder". This sequence of events implements the outorder logic since their arrival until they are finished.

. Event no. 10 - Arrival of an outorder

Event no. 10 is scheduled with the outorder control entity (ORDCON), and the first time it is executed corresponds to the first outorder to arrive and it is scheduled by program IDUMP (see 5.3.3.1.5.). Event no. 10 starts by checking the despatch bays waiting queue. If no despatch bays are available then it just updates the ORDCON attribute number of outorders waiting. Otherwise: it takes the first despatch bay from the

waiting queue; adds it to the despatch bays processing queue; takes an outorder entity from the outorders neutral queue and defines its attributes - time of arrival and pointer to the despatch bay entity; and schedules event no. 11 with the despatch bay entity. Finally, event no. 10 schedules (itself) the arrival of the next outorder with the ORDCON entity.

. Event no. 11 - Outorder initialisation

Event no. 11 is scheduled with the despatch bay entity. It starts by reading from the outorders file the inter-arrival time, the order number and the number of order lines. It then defines the outorder attributes: order number and time of processing start.

Event no. 11 starts a loop for each line in the outorder by reading the product code, number of pallets and the number of cases. If the line includes pallets then event no. 11 checks the product cells pointer vector, and allocates product pallets until it reaches the number of pallets asked for or there is an out of stock situation. For each product cell with pallets allocated: it takes a pallet entity from the neutral queue; defines the pallet attributes (pointer to the cell and number of pallets to fetch); and adds the pallet entity to the outorder pallets waiting queue. If there is an out of stock situation it updates the product attributes: number of stock out conditions, and number of pallets out of stock. If an outorder line asks for product cases event no. 11, follows a logic equivalent to the one used for the pallets

Several outorder entity attributes are defined (e.g. no. of pallets, no. of cases, no. of pallets out of stock, no. of cases out of stock, no. of pallets yet to satisfy, and no. of cases yet to satisfy). Subroutine DISTDE is called to display the truck icon at the despatch bay. Finally, if there are pallets or cases to satisfy, event no. 12 is scheduled, otherwise, the outorder finishes and so event no. 13 is scheduled instead.

. Event no. 12 - Start processing an outorder

Event no. 12 is scheduled with the despatch bay entity. The job type no. 2 entity is used to access the job priority queues. A loop is made to inspect the job priority queue by descending order of priority. If a transporter is available, subroutine DPATRD is called to allocate a pallet to it. The same logic is used for the cases but the job type no. 4 entity is used to access the job priority queues and subroutine DCATRD is called to try to allocate cases to the transporter.

. Event no. 13 - Finish processing an outorder

Event no. 13 is scheduled with the despatch bay entity. It resets some transporter and despatch bay attributes, and updates the outorders statistics. If there is an outorder waiting, it follows an event no. 10 similar procedure and then schedules the outorder initialisation event no. 11. Otherwise, it takes the despatch bay from the processing queue and adds it to the despatch bays neutral queue.

5.3.4.3.5. Loading Job

This group of events includes event no. 14 "Start transporter job no. 2", event no. 15 "Load a pallet from reserve storage", event no. 16 "Move a pallet to a despatch bay" and event no. 17 "Unload a pallet to a despatch bay". This sequence of events implements the logic for transporter job no. 2 which consists of a transporter loading a pallet at reserve sorage and taking it to a despatch bay. All the events in this sequence are scheduled with the job entity.

. Event no. 14 - Start transporter job no. 2

Event no. 14 calls subroutine CEACCO to compute the cell access coordinates. Then it calls subroutine DEFPAT to define the transporter path, between the current transporter position and the cell, and to schedule the start of the transporter movement. Subroutine DEFPAT checks if the initial and final path positions can be taken as being the same. If they are the same, it schedules event (15) passed as a parameter by the calling subroutine. Otherwise it calls subroutine AUXPAT which computes the individual paths, adds the transporter to the movement queue, defines the transporter attributes, calls subroutine COPTRA and schedules event no. 9 for the current simulation time plus the transporter type attribute - time for start and stop.

. Event no. 15 - Load a pallet from reserve storage

Event no. 15 starts by checking if the cell is part of a PMR (Powered Mobile Racking) type of racking. In this case for technical reasons (see fig. 5.34) just one transporter can access the racking at the same time. If the racking is PMR and is already being accessed then the transporter is added to a waiting queue (PMR waiting transporters vector) and

the event no. 15 ends. Otherwise it follows the instructions common to the other type of rackings.

Event no. 15 calls subroutine CELCOR to calculate the cell coordinates. Then it reads the transporter type attributes to define the transporter performance characteristics. It calculates the time taken by the transporter to reach the cell, it moves up, load the pallet and moves down again. This time is used to schedule event no. 16 to take the pallet to the despatch bay. Event no. 15 also updates the transporter position on the screen, perpendicular to the racking, indicating that the transporter is going to the cell and is loading the pallet.

. Event no. 16 - Move a pallet to a despatch bay

Event no. 16 checks if the cell is part of a PMR (Powered Mobile Racking) type of racking and if it is, calls subroutine JOBPMR to update the PMR transporter waiting queue. Subroutine JOBPMR updates the waiting queue and if there is a transporter waiting to access the racking, schedules the appropriate event. Event no. 16 calls subroutine DEFPAT to define de transporter path to the despatch bay and to schedule the beginning of the transporter movement.

Event no. 16 then updates the number of cell occupied slots and calls subroutine DISPAC to update the cell pallets on the screen display. Update the product and zone attributes number of pallets. It checks the product group attribute - minimum number of reserve pallets, and if necessary generates an inorder sheduling event no. 30 (inorder arrival) for the current time plus the time delay for inorder arrival. If the cell slots become empty it calls subroutine TACEPR to withdraw the cell from the product and add it to the sorted zone free cells queue. The cell attribute, sort key number, defined during the configuration stage, is used as a key to sort the cells in the queue.

. Event no. 17 - Unload a pallet to a despatch bay

Event no. 17 reads the transporter type attribute, fixed time for unloading at the despatch bay, and uses this value to schedule the next event. Event no. 17 updates the outorder attributes, number of pallets, and number of cases yet to satisfy. If the outorder is finished then it schedules event no. 13 (finish processing an outorder). It takes the job entity from the jobs processing queue, and adds it to the jobs neutral queue.

Event no. 17 checks the transporter jobs waiting queue and, if there are any jobs there, schedules event no. 8 (start a transporter job) with the first job entity waiting in the queue, otherwise it schedules event no. 7 (free transporter) with the transporter entity. Finally, it calls subroutine SCADET to update the truck icon at the despatch bay, on the screen display, with a graphic bar proportional to the amount of the outorder already satisfied.

5.3.4.3.6. Picking Job

This group of events includes event no. 20 "Start transporter job no. 4", event no. 21 "Start picking cases", event no. 22 "Finish picking cases", event no. 23 "Move cases to a despatch bay" and event no. 24 "Unload cases at a despatch bay". This sequence of events implements the logic for transporter job no. 4, which consists of a transporter loading cases at picking cells, and when the pallet is full, taking the pallet to a despatch bay. All the events in this sequence are scheduled with the job entity.

. Event no. 20 - Start transporter job no. 4

Event no. 20 is similar to event no. 14. It starts by calling subroutine CEACCO, to compute the access cell coordinates, and then calls subroutine DEFPAT to define the transporter path. When the transporter arrives to the cell, event no. 21 (Start picking cases) is scheduled.

. Event no. 21 - Start picking cases

Event no. 21 is similar to event no. 15. There is a slight difference in the calculation of the time taken by the transporter to reach the cell and load the cases. The difference has to do with the time taken to load the cases, which depends on the number of cases to load and on the transporter type attribute - number of cases picked per minute. Also, the next event to be schedule is event no. 22 (Finish picking cases).

. Event no. 22 - Finish picking cases

Event no. 22, in the same way as event no. 16, checks if the cell belongs to a PMR type of racking, and if it is true, calls subroutine JOBPMR to update the PMR transporter

waiting queue. It updates the number of cases in the pallet at the picking cell. Then it calculates the minimum number of cases under which it is necessary to replenish the picking cell. This value is equal to the transporter attribute - number of cases per pallet, times the product group attribute - proportion of pallet re-order level. If the current number of cases at the picking cell is less than this value, and no replenishment order is pending, then a replenishment order is issued.

The replenishment can be processed in two ways: automatically, if the picking cell has more then one pallet (e.g. live storage racking) the number of pallets in the picking zone is updated (less one) and the number of cases at the picking cell is increased by the number of cases per pallet for the product; or a replenishment order is issued. If a replenishment order is issued: the product entity is added to the products waiting for replenishment queue (if it was not already there); subroutine CHEFCE is called to see if there are any reserve pallets of the product; and job no. 4 priority queues are inspected, and subroutine DERTCE is called to try to allocate a pallet to a transporter.

If necessary, subroutine DISPAC is called to update the number of cell pallets on the screen display. If the picking list is not finished, event no. 20 (start transporter job no. 4) is scheduled, otherwise it means that the pallet is full, or the picking list ended and on both cases event no. 23 is scheduled for the pallet to be taken to the despatch bay.

. Event no. 23 - Move cases to a despatch bay

Event no. 23 just calls subroutine DEFPAT to define the transporter path to the despatch bay, and to schedule the beginning of the transporter movement.

5.3.4.3.7. Replenishment job

This group of events includes event no. 25 "Start transporter job no. 3", event no. 26 "Load pallet from reserve storage", event no. 27 "Take a pallet to picking storage for replenishment", event no. 28 "Start replenishment from pallet" and event no. 29 "Finish replenishment from pallet". This sequence of events implements the logic for transporter job no. 3, which consists of a transporter loading a product pallet at reserve storage and takes it to the picking cell. All the events in this sequence are scheduled with the job entity.

. Event no. 25 - Start transporter job no. 3

Event no. 25 is similar to event no. 14. It starts by calling subroutine CEACCO, to compute the access cell coordinates, and then calls subroutine DEFPAT to define the transporter path. When the transporter arrives at the cell, event no. 26 (Load a pallet from reserve storage) is scheduled.

. Event no. 26 - Load a pallet from reserve storage

Event no. 26 is similar to event no. 15 except that the next event to be scheduled is event no. 27 (Take a pallet to picking storage for replenishment).

. Event no. 27 - Take a pallet to picking storage for replenishment

Event no. 27 is similar to event no. 16. Subroutine CEACCO is called to compute the cell coordinates corresponding to the final position of the travel path. Subroutine DEFPAT is then called to define the transporter path to the picking cell, and to schedule the beginning of the transporter movement. The next event to be scheduled, passed as parameter to subroutine DEFPAT, is event no. 28 (Start replenishment from pallet).

. Event no. 28 - Start replenishment from pallet

Event no. 28 calls subroutine CELCOR to calculate the picking cell coordinates. Then it updates the transporter position on the screen display and schedules event no. 29 (Finish replenishment from pallet)

. Event no. 29 - Finish replenishment from pallet

Event no. 29 updates the picking cell attributes: number of pallets in the cell (+1); number of pallets asked for (-1); and number of pallets expected (-1). It updates the number of pallets in the picking zone, and calls subroutine DISPAC to update the number of picking cell pallets on the screen. It increases the number of cases, at the picking cell, by the product number of cases per pallet. Event no. 29 checks the transporter jobs waiting queue and, if there are any jobs, schedules event no. 8 (start a transporter job) with the first job entity waiting in the queue, otherwise it schedules event no. 7 (free transporter) with the transporter entity.

5.3.4.3.8. Unloading job

This group of events includes event no. 32 "Start transporter job no. 1", event no. 33 "Load a pallet from the reception bay", event no. 34 "Move a pallet to a reserve storage location", event no. 35 "Start unloading a pallet to a reserve storage location" and event no. 36 "Finish unloading a pallet to a reserve storage location". This sequence of events implements the logic for transporter job no. 1 which consists of a transporter loading a product pallet at a reception bay, and taking it to reserve storage. All the events in this sequence are scheduled with the job entity.

. Event no. 32 - Start transporter job no. 1

Event no. 32 just calls subroutine DEFPAT to define the transporter path to the reception bay and to schedule the beginning of the transporter movement. The next event to be scheduled, passed as parameter to subroutine DEFPAT, is event no. 33 (Load a pallet from the reception bay).

. Event no. 33 - Load a pallet from the reception bay

Event no. 33 schedules event no. 34 (Move a pallet to a reserve storage location) for the current simulation time plus the transporter type attribute - fixed time for loading in the reception bay.

. Event no. 34 - Move a pallet to a reserve storage location

Event no. 34 starts by updating the inorder attribute - number of pallets remaining. If there are no more pallets remaining and no more pallets waiting for a cell, then it schedules event no. 37 (Inorder end). It calls subroutine CEACCO to calculate the cell coordinates and calls subroutine DEFPAT to define the transporter path to the cell and to schedule the beginning of the transporter movement. The next event to be scheduled, passed as parameter to subroutine DEFPAT, is event no. 35 (Start unloading a pallet to a reserve storage location). Finally, event no. 34 calls subroutine SCARET to update the truck icon at the reception bay, on the screen display, with a graphic bar proportional to the amount of the inorder yet to unload. The colour used to draw the graphic bar is the product group attribute - colour for display.

. Event no. 35 - Start unloading a pallet to a reserve storage location

Event no. 35 is similar to event no. 15, except in the calculation of the time to access the cell and in the next event to be scheduled which is event no. 36 (Finish unloading a pallet to a reserve storage location).

. Event no. 36 - Finish unloading a pallet to a reserve storage location

Event no. 36, in the same way as event no. 16, checks if the cell belongs to a PMR type of racking and if it does, calls subroutine JOBPMR to update the PMR transporter waiting queue. It updates the cell attributes: number of pallets in the cell (+1); and number of pallets coming to (-1). Then it updates the number of pallets in the racking zone. If the cell is not already assigned to the product, it calls subroutine ADCEPR to add the cell to the product cell pointers vector. This vector stores the cells sorted by the cell attribute - sort key number. This number depends not only on the cell priority, defined during the configuration stage, but also on the product reserve storage zone (first, second or third) where it is located.

Event no. 36 updates the product group attribute - number of pallets. Then it calls subroutine DISPAC to update the number of cell pallets on the screen display and takes the job entity from the jobs processing queue and add it to the jobs neutral queue. Finally, event no. 36 checks the transporter jobs waiting queue and, if there are any jobs, schedules event no. 8 (start a transporter job) with the first job entity waiting in the queue, otherwise it schedules event no. 7 (free transporter) with the transporter entity.

5.3.4.3.9. Inorders

This group of events includes event no. 30 "Inorder arrival", event no. 31 "Inorder initialisation" and event no. 37 "Inorder end". This sequence of events implements the logic for the arrival, initialisation and inorders end.

. Event no. 30 - Inorder arrival

Event no. 30 is scheduled with the inorder control entity (INOCON). It starts by checking the reception bays waiting queue. If no reception bays are available then it just updates the INOCON attribute - number of inorders waiting, and adds the product to the inorder products waiting queue. Otherwise: it takes the first reception bay entity from the reception bays waiting queue; adds it to the reception bays processing queue; takes an inorder entity from the inorders neutral queue, and defines its attribute - pointer to the reception bay entity; and schedules event no. 31 with the reception bay entity.

. Event no. 31 - Inorder initialisation

Event no. 31 is scheduled with the reception bay entity. It starts by defining the inorder attributes: time of arrival; time of processing start; total number of pallets; number of pallets waiting for cell; and number of pallets remaining. Then, using the product cell pointers vector, it checks the cells assigned to the product for free slots, and allocates pallets to these.

After this stage, if there are still pallets, it goes through the product reserve storage zones free cells queue and if there are cells available, it assigns them to the product and allocates pallets to the cell slots. The allocation of a pallet to a cell slot is made using the following procedure: the cell attribute - number of pallets expected is updated (+1); a pallet entity is taken from the pallets neutral queue; the pallet attribute - pointer to the cell is defined; and the pallet entity is added to the inorder pallets waiting queue.

If all the pallets are not allocated in this two stages then it is updated, the inorder attribute - number of pallets waiting for cell. Subroutine DPAREC is called to try to allocate available transporters to do the unloading job. Subroutine DISTRE is called to display the truck icon at the reception bay.

. Event no. 37 - Inorder end

Event no. 37 is scheduled with the inorder entity. It resets some transporters and reception bay attributes, and updates the inorders statistics. If there is an inorder waiting, it follows an event no. 30 similar procedure, and then schedules the inorder initialisation event no. 31. Otherwise it takes the reception bay from the processing queue and adds it to the reception bays neutral queue.

5.3.4.3.10. Parking

This group of events includes event no. 18 "Return a transporter to park" and event no. 19 "Park a transporter". This sequence of events implements the logic for the returning of a transporter to park when idle. All the events in this sequence are scheduled with the job entity.

. Event no. 18 - Return a transporter to park

Event no. 18 just calls subroutine DEFPAT to define the transporter path to the parking location and to schedule the beginning of the transporter movement. The next event to be scheduled, passed as parameter to subroutine DEFPAT, is event no. 19 (Park a transporter).

. Event no. 19 - Park a transporter

Event no. 19 takes the job entity from the jobs processing queue and add it to the jobs neutral queue. If the transporter breakdown flag attribute is ON (it was set during the returning to park) event no. 19: updates the transporter idle statistics; defines the transporter attribute - occupation flag = 0; and takes the transporter from the processing queue. Otherwise, event no. 19: defines the transporter attribute - occupation flag = -1; and schedules event no. 7 (Free transporter) with the transporter entity. Event no. 7 is scheduled to check if during the return of the transporter to park, new tasks have been generated.

5.3.4.3.11. Breakdown

This group of events includes event no. 38 "Return transporter to park breakdown / end of activity" and event no. 39 "Transporter in the park breakdown / end of activity". This sequence of events implements the logic for the returning of a transporter to park when it is broken down or when it retires from activity at the end of a shift. All the events in this sequence are scheduled with the job entity.

. Event no. 38 - Return transporter to park breakdown / end of activity

Event no. 38 just calls subroutine DEFPAT to define the transporter path to the parking location and to schedule the beginning of the transporter movement. The next event to be scheduled, passed as parameter to subroutine DEFPAT, is event no. 39 (Transporter in the park breakdown / end of activity).

. Event no. 39 - Transporter in the park breakdown / end of activity

Event no. 39 takes the job entity from the jobs processing queue and adds it to the jobs neutral queue. If the transporter breakdown flag attribute is OFF (it was cleared during the return to park) event no. 39: updates the transporter breakdown statistics; defines the transporter attribute - occupation flag = -1; and schedules event no. 7 (Free transporter) with the transporter entity. Otherwise, event no. 19: defines the transporter attribute - breakdown flag = 1; and takes the transporter from the processing queue.

5.3.4.3.12. Shifts

This group of events includes event no. 40 "Transporter shift starts", event no. 41 "Transporter shift ends", event no. 42 "Bay shift starts" and event no. 43 "bay shift ends". These events implement the logic for the transporter and bays entering and retiring from activity. These events are scheduled with the virtual transporter, despatch and reception entities.

. Event no. 40 - Transporter shift starts

Event no. 40 defines the transporter attribute - activity flag (=1). If the transporter is idle at the park then: it defines the transporter attributes - initial operation time and initial activity time; deletes the inactivity mark from the screen display; and if the transporter is not broken down it adds the transporter to the processing queue and schedules event no. 7 (Free transporter). Finally, it schedules event no. 41 (Transporter shift ends).

. Event no. 41 - Transporter shift ends

Event no. 41 defines the transporter attribute - activity flag (=0). If the transporter is idle at the park then: it defines the transporter attributes - time idle and time in activity;

shows the inactivity mark on the screen display; and takes the transporter from the jobs priorities queues. Otherwise if the transporter is broken down then it updates the transporter breakdown statistics, and shows the inactivity mark on the screen display. Finally, it schedules the event no. 40 (Transporter shift starts).

. Event no. 42 - Bay shift starts

Event no. 40 defines the (reception / despatch) bay attribute - activity flag (=1). It takes the (reception / despatch) bay from the inactivity queue. If the bay was idle when it retired from activity then it adds it to the (reception / despatch) bays neutral queue. Otherwise, if it is a reception bay it follows a procedure similar to event no. 30 (Inorder arrival) and schedules event no. 31 (Inorder initialization); else if it is a despatch bay, it follows a procedure similar to event no. 10 (Arrival of an outorder) and schedules event no. 11 (Outorder initialization). It changes the inactivity mark to the acivity mark at the bay on the screen display. Finally, it schedules event no. 43 (Bay shift ends)

. Event no. 43 - Bay shift ends

Event no. 43 defines the (reception / despatch) bay attribute - activity flag (=0). If the (reception / despatch) bay is idle then: it takes the (reception / despatch) bay from the neutral queue, and adds it to the inactivity queue, and changes the activity mark to the inactivity mark at the bay on the screen display. Then it schedules event no. 42 (Bay shift starts).

5.3.4.3.13. Reports

This group of events includes only one event event no. 44 "Generate a report and schedule the next one". Event no. 44 generates a printed report to a file of the current warehouse performance statistics. This event is scheduled with the report entity.

. Event no. 44 - Generate a report and schedule the next one

Event no. 44 just calls the REPORT subroutine, to print out the report, and then it schedules (itself) event no. 44 (Generate a report and schedule the next one) for the current simulation time plus the report entity attribute - report time interval. Figure 5.66 shows the warehouse performance report which is printed at regular intervals.

APPLICATION NAME: DEMO Page 1

Dump number: 10

Start date: 09 JUL 90 08:00 Current date: 09 JUL 90 08:00

Duration: 0 hours, 0 minutes, 0 seconds

INPUT:

Loads arrival: 0 Pallets input: 0

Average waiting time: 0.0 min Average processing time: 0.0 min

OUTPUT:

Orders despatched: 0

Pallets output: 0 Cases output: 0

Out of stock pallets: 0 Out of stock cases: 0

Average waiting time: 0.0 min Average processing time: 0.0 min

ZONE INFORMATION

Zone Name	Number of Pallets	Pallets Positions	Percentage Fill
LIVE	210	240	87.5
DRIVE	85	340	25.0
APR/PICK	60	88	68.2
APR/SD	110	110	100.0
APR/DD	150	1216	12.3

PRODUCT GROUP INFORMATION

Product Group	Products	Pallets	Picking Failures Qnty	Pallets	Qnty	Cases
PRODUGR1	7	105	0	0	0	0
PRODUGR2	7	105	0	0	0	0
PRODUGR3	7	105	0	0	0	0
PRODUGR4	6	60	0	0	0	0
PRODUGR5	6	60	0	0	0	0
PRODUGR6	6	60	0	0	0	0

TRANSPORTER INFORMATION

Type Name	Qnty	Active	Unload	Load	Replen.	Pick.	Wait.	Idle	Breakd.
PICK TRUCK	2	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
REACH TRUC	2	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
MANUAL	2	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
STOCA	1	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Figure 5.66 The warehouse performance report generated at regular time intervals

CHAPTER 6

EXAMPLES

6. Examples.

6.1. Introduction

In this chapter some examples of using the AWARD system are presented. The first example "DEMO" has been used during the development stages to verify and validate the AWARD system and to demonstrate its potential to people from the warehousing and other business.

The second example "LUTTER" is a model of a warehouse managed by one of the consortium members (NCCS - National Carriers Contract services). "LUTTER" was been the first model developed to simulate an existing warehouse. Its development and use involved people working in the warehouse which have given a very important contribution for the validation process and for the specification of future developments.

AWARD system has been used by students from Cranfield MSc in logistics and distribution. The third example is taken from one MSc Thesis [Jamani 1989]. The objective of this work was setting up a model of the main warehouse of one of the consortium members, a major electrical manufacturer and supplier, in order to evaluate the AWARD system.

The fourth example concerns the use of the AWARD system in a consultancy work which involved the computer simulation of a large soft drinks warehouse [Cooper & Saw 1990]. The AWARD system was used to evaluate if the proposed warehouse configuration and system logic were capable of handling the work load through the peak and normal operation.

6.2. Example 1 - DEMO.

This example has been used to verify and validate the AWARD system. The "DEMO" model has been run exhaustively with many different parameters. The "DEMO" warehouse configuration has a reasonable size and it has most of the features found in most common warehouses. The warehouse throughput includes full pallets and case picking.

6.2.1. Warehouse model description.

The "DEMO" warehouse, as it has been used to verify and validate the AWARD system, has a configuration which includes most of the technological options available in AWARD. The "DEMO" warehouse layout is shown in Figure 6.1 where the main warehouse areas (No-go areas, Pillars, Despatch and Reception areas), the type of racking and the transporter parking positions are highlighted. The warehouse has three despatch bays and two reception bays.

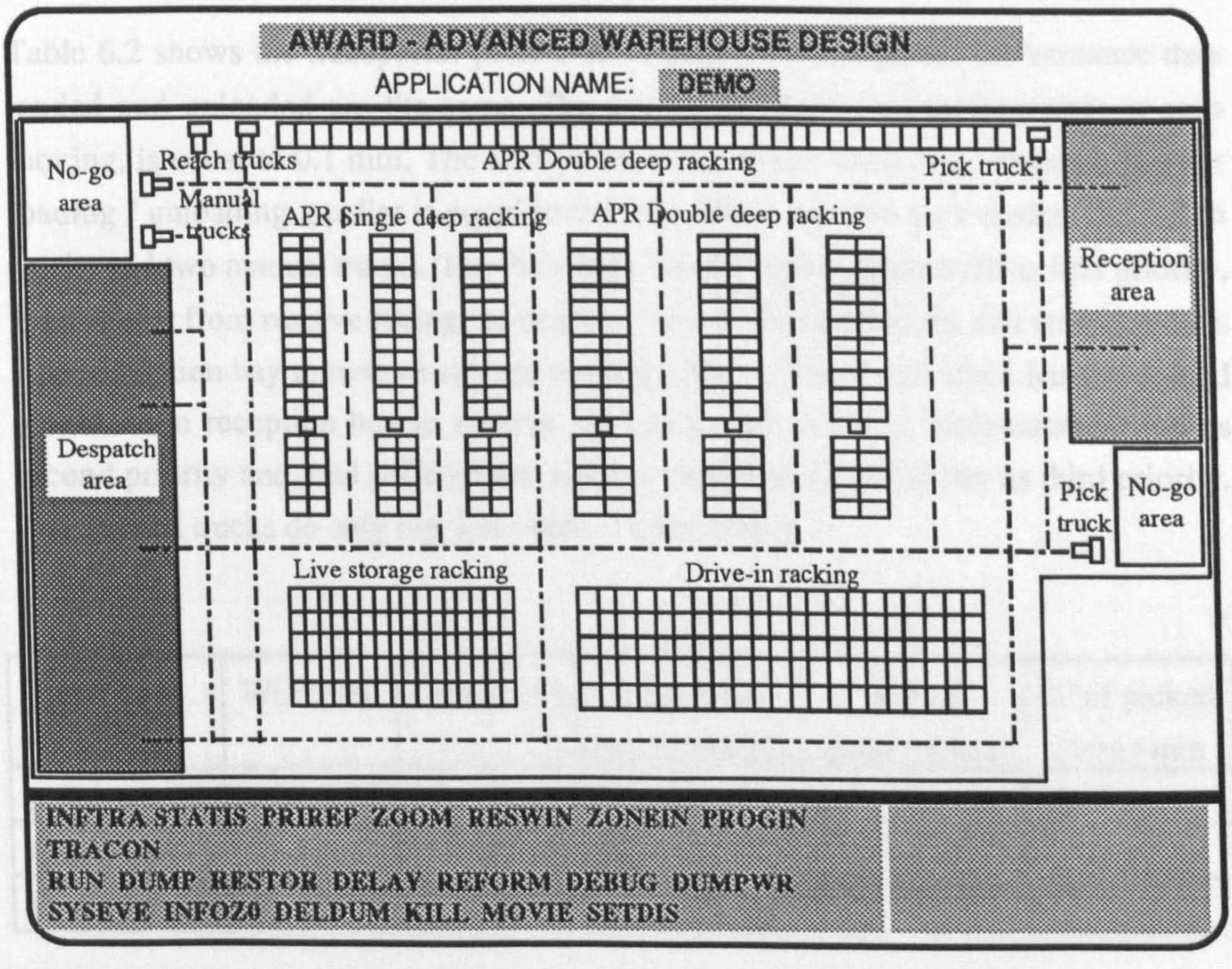


Figure 6.1 The warehouse configuration used in example 1 - DEMO

Table 6.1 shows the number of levels and cells for each racking type and the racking module dimensions. The warehouse has a total of 1412 cells.

Racking	Nº of levels	Nº of cells	Module width (m)	Module depth (m)	Module height (m)
Live storage	2	160	1.5	2.0	1.5
Drive-in	4	420	1.6	2.0	1.5
APR/SD	4	288	1.7	2.0	1.5
APR/DD	3	544	1.7	2.8	1.5

Table 6.1 Racking data from example 1 - DEMO

Table 6.2 shows the transporter performance data. The transporter performance data loaded and unloaded are the same. The transporter delay, when they start or stop moving, is equal to 0.1 min. The transporter delay at the reception / despatch bay for loading / unloading a pallet is equal to 0.5 min. There are two pick trucks, two reach trucks and two manual trucks. The Pick truck has the replenishment job as first priority, load pallets from reserve storage to despatch bay as second priority and unload pallets from reception bay to reserve storage as third priority. The Reach truck has the unload pallets from reception bay to reserve storage as first priority, replenishment job as second priority and load pallets from reserve storage to despatch bay as third priority. The manual trucks do only one job which is case picking.

Name	Width (m)	Length (m)	Horizontal speed (m/min)	Vertical speed (m/min)	Nº of picked cases / min
Pick truck	1.5	2	120	40	60
Reach truck	1.5	2	120	40	0
Manual truck	1.4	1.8	90	40	60

Table 6.2 Transporter performance data from example 1 - DEMO

There are five racking zones called LIVE, DRIVE, APR/PICK, APR/SD and APR/DD. The LIVE zone includes the cells from the live storage racking. The DRIVE zone includes the cells from the drive-in racking. The APR/PICK includes the cells from the

first level of the first APR/SD racking on the left (see Figure 6.1). The APR/SD zone includes the remaining cells from the APR/SD racking. The APR/DD zone includes the cells from the APR/DD racking.

Table 6.3 presents the product group characteristics. For a detailed description of the items from the first column see 5.3.2.3.. The product group Standard Deviation of the nº of cases / pallet is equal zero which means that all the products within a group have the same nº of cases per pallet and equal to the average nº of cases / pallet. The time delay for inordering is equal zero which means that when the number of pallets for a product goes below the minimum number of reserve pallets, then an inorder is generated for that product which will arrive immediately. The number of inorder pallets is defined by the product group EOQ value. The product group colour index is used to distinguish the product pallets in the warehouse display and for identifying the product groups in the bar charts.

	Product group number							
	1	2	3	4	5	6	7	8
Nº of products	7	7	7	6	6	6	6	6
Av.nºcas./pallet	30	30	40	30	30	30	30	30
SD nºcas./pallet	0	0	0	0	0	0	0	0
Max.res.pallets	30	30	30	30	30	30	30	30
Min.res.pallets	5	5	5	8	8	8	8	8
1º storage zone	drive	drive	drive	apr/sd	apr/sd	apr/sd	apr/sd	apr/sd
2º storage zone	apr/dd	apr/dd	apr/dd	apr/dd	apr/dd	apr/dd	apr/dd	apr/dd
3º storage zone								
Picking zone	live	live	live	apr/pic	apr/pic	apr/pic	apr/pic	apr/pic
Pallet ROL (%)	50	50	50	50	50	50	50	50
Colour index	1	2	3	4	5	6	7	8
EOQ for pallets	10	10	10	20	20	20	20	20
Time delay	0	0	0	0	0	0	0	0

Table 6.3 Product group data from example 1 - DEMO

Figure 6.2 represents a typical outorders arrival path, and the number of cases and pallets of each outorder. The model has been run with several outorder files generated by the OUTORDER program.

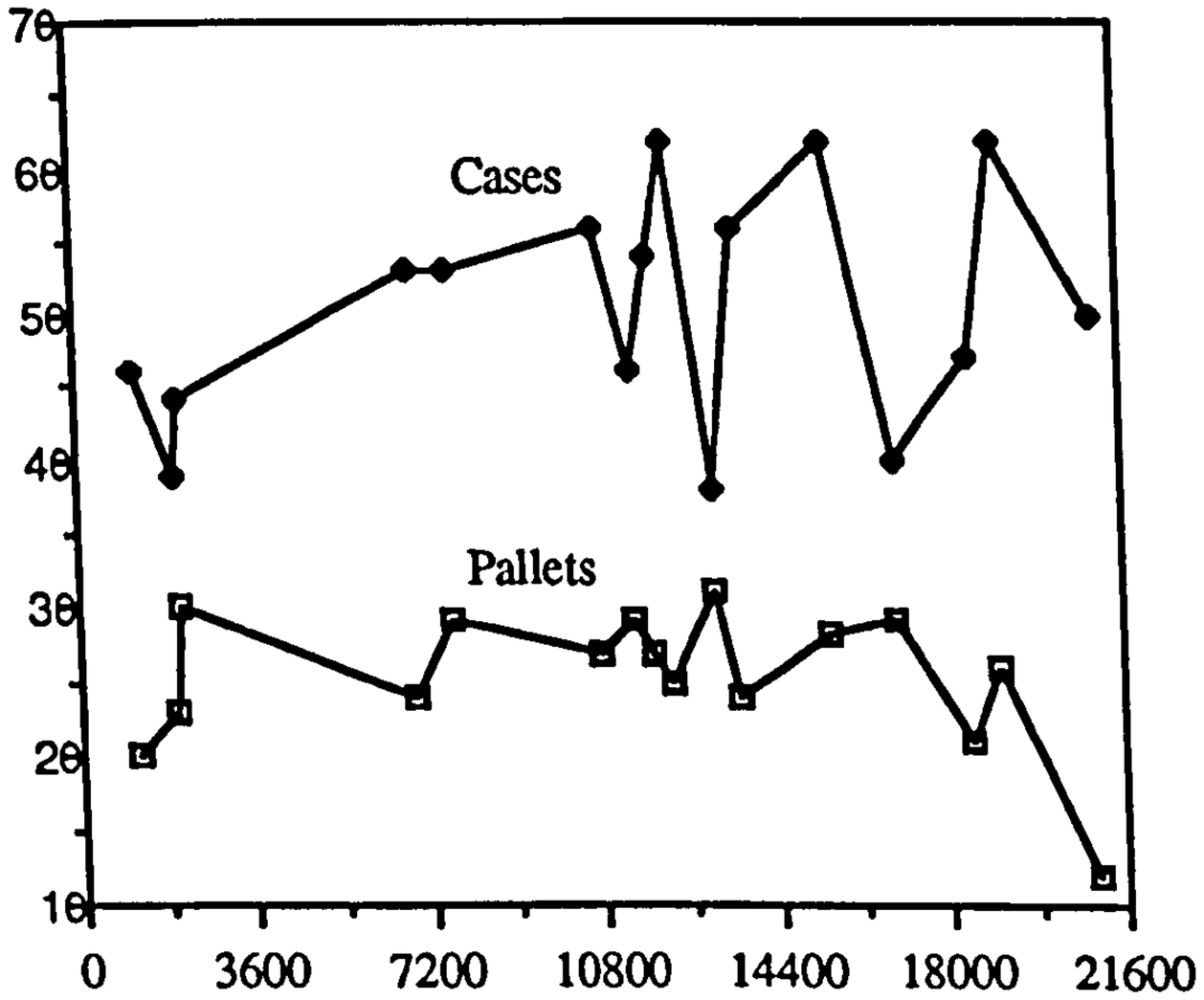


Figure 6.2 A typical outorders arrival path

Figure 6.3 shows a picture from the simulation screen display during the running of the "DEMO" model. Figure 6.4 shows bar charts with the transporter performance data obtained using interaction INFTRA.

6.2.2. Conclusions.

This example has been used during the development stages to correct software problems and to do the tuning of the model logic. The fact that it has been run many times and for long periods of simulation time has increased confidence in the system. The use of this model to demonstrate the AWARD capabilities has also been very valuable for convincing people of the advantages of this approach.

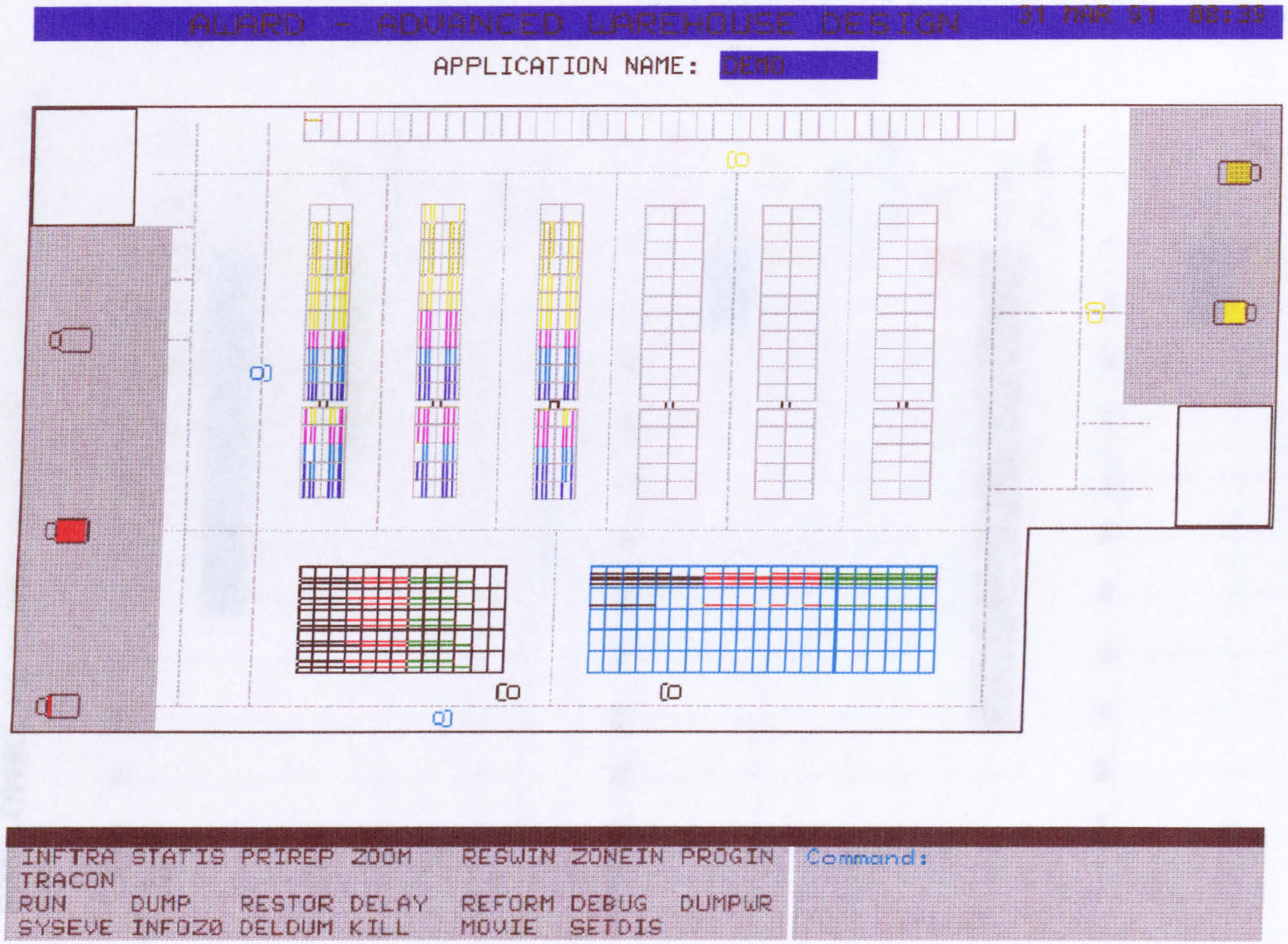
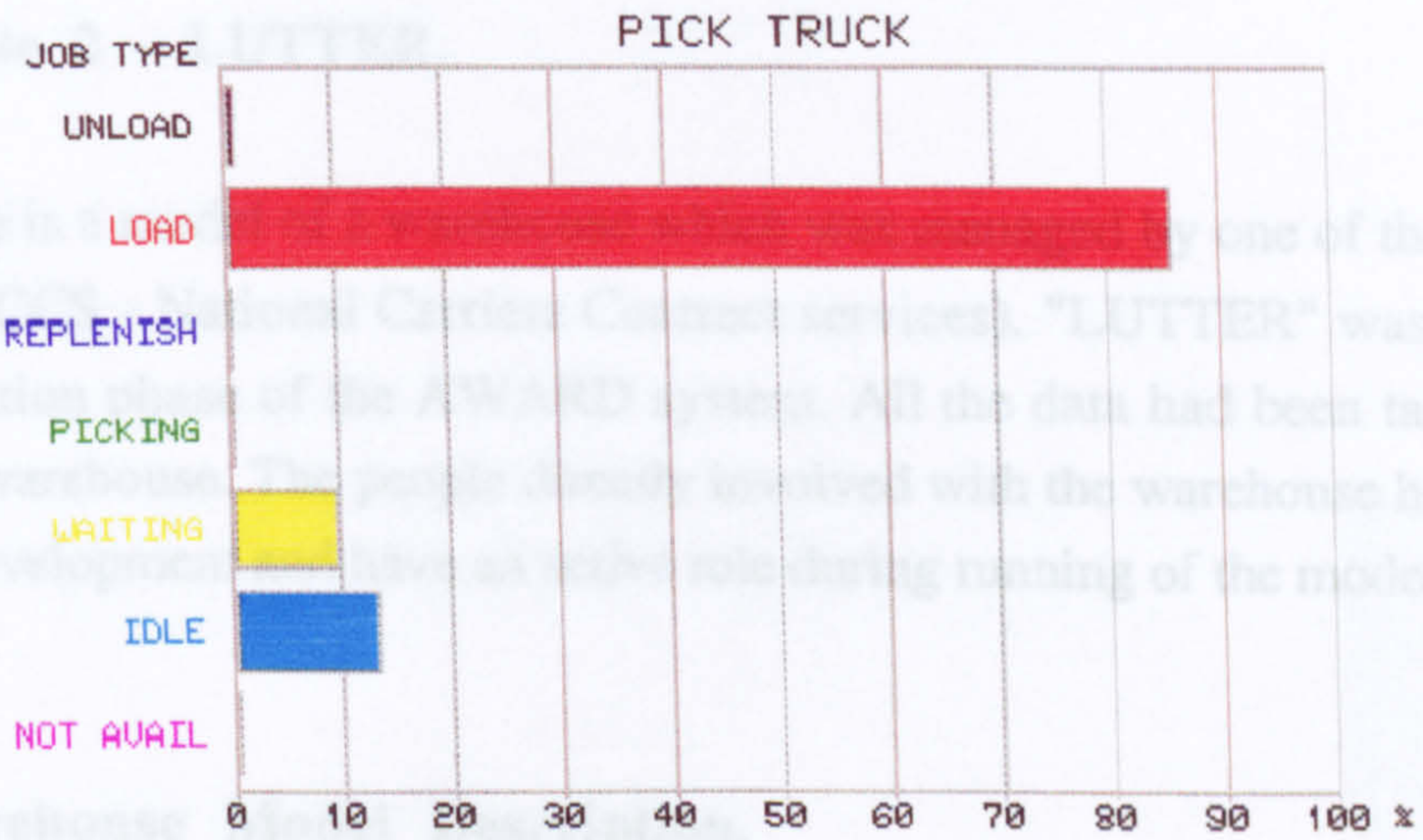
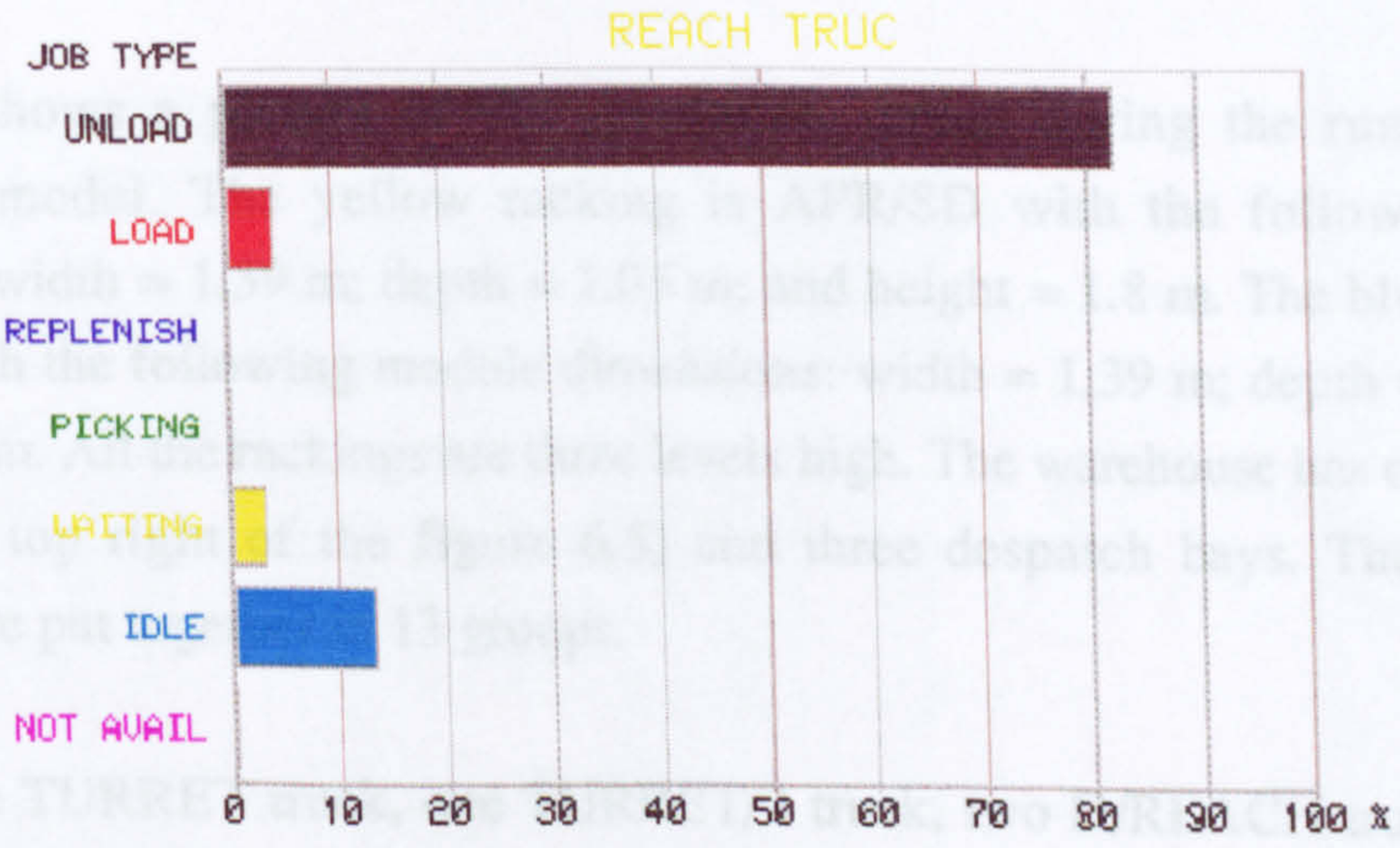


Figure 6.3 Screen display during the running of the "DEMO" model





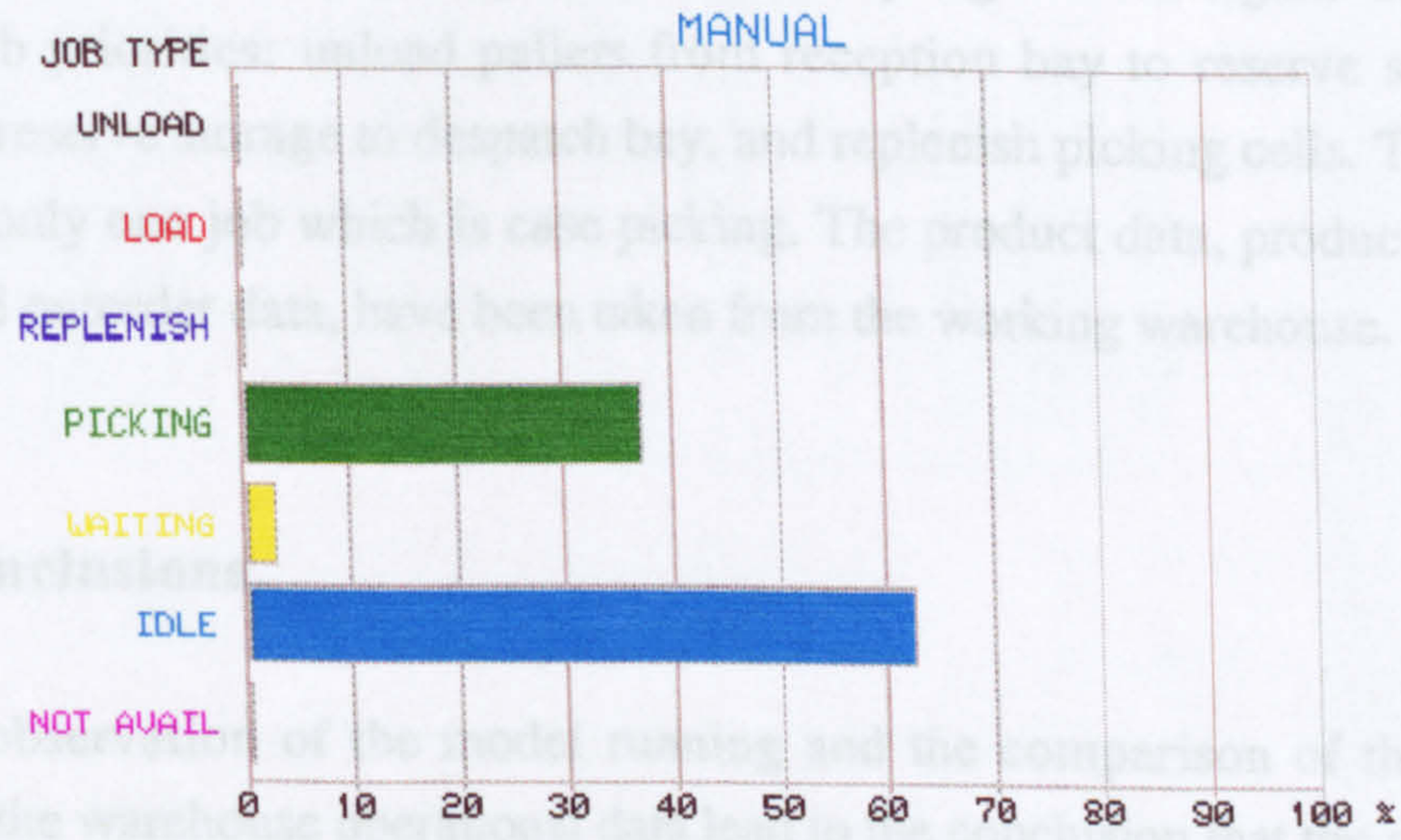


Figure 6.4 Transporter performance charts for the "DEMO" model

6.3. Example 2 - LUTTER.

This example is a model of a warehouse which was managed by one of the consortium members (NCCS - National Carriers Contract services). "LUTTER" was used as part of the validation phase of the AWARD system. All the data had been taken from the operational warehouse. The people directly involved with the warehouse have helped in the model development and have an active role during running of the model.

6.3.1. Warehouse Model Description.

Figure 6.5 shows a picture of the simulation screen during the running of the "LUTTER" model. The yellow racking is APR/SD with the following module dimensions: width = 1.39 m; depth = 1.05 m; and height = 1.8 m. The blue racking is APR/DD with the following module dimensions: width = 1.39 m; depth = 2.1 m; and height = 1.8 m. All the rackings are three levels high. The warehouse has one reception bay (on the top right of the figure 6.5) and three despatch bays. The warehouse products were put together in 13 groups.

There is one TURRET truck, one TURRET/2 truck, two D/REACH trucks and five PALLET trucks. The first four (parked at the top right from figure 6.5) have the following job priorities: unload pallets from reception bay to reserve storage; load pallets from reserve storage to despatch bay; and replenish picking cells. The PALLET trucks have only one job which is case picking. The product data, product zoning, job priorities and outorder data, have been taken from the working warehouse.

6.3.2. Conclusions.

The direct observation of the model running and the comparison of the simulation results with the warehouse operational data lead to the conclusion that the model had the required level of accuracy. The direct involvement of the warehouse people has been very important for the validation process. Some of model adjustments have resulted from the direct observation of the simulation screen together with the experience of how the warehouse operation works in reality.

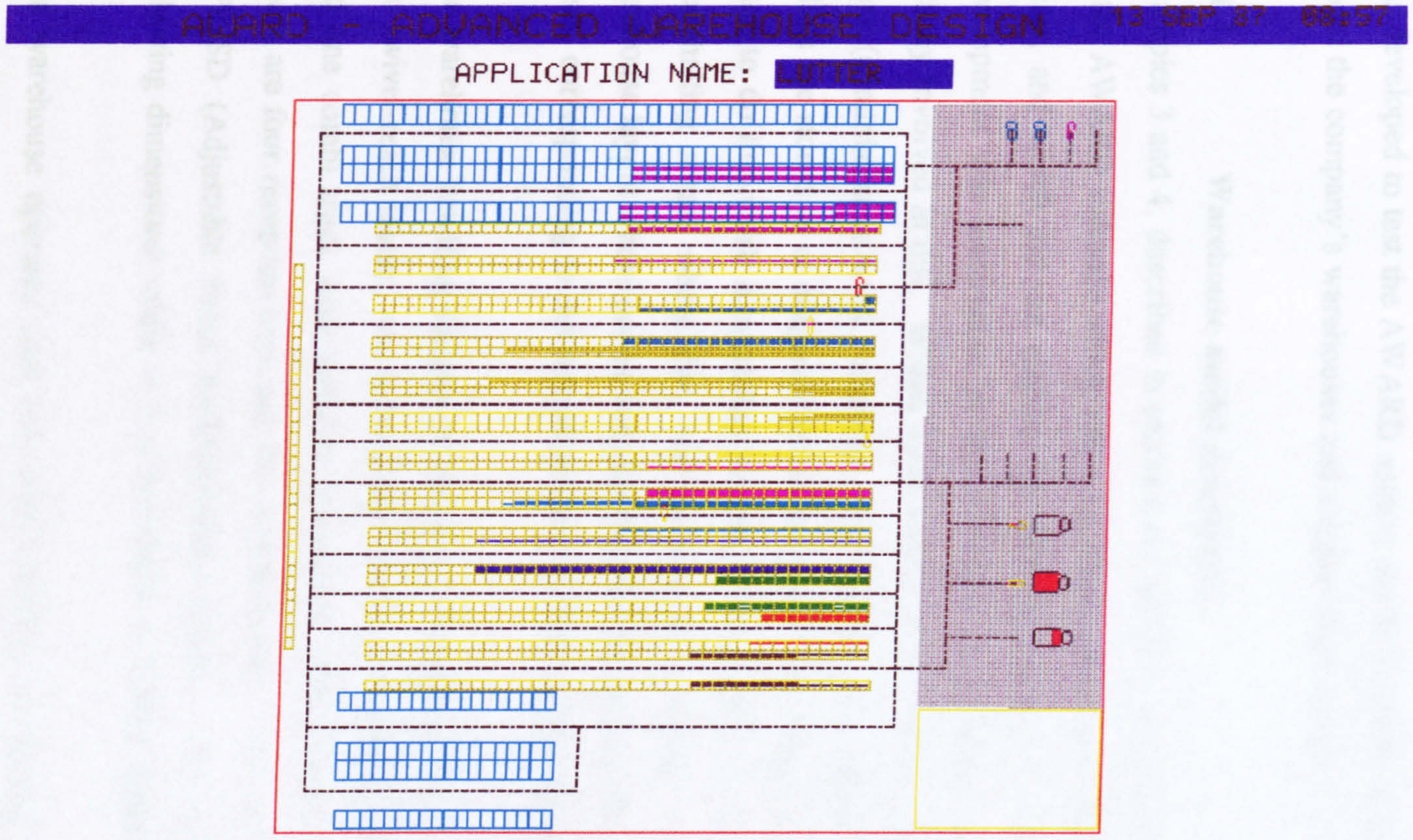


Figure 6.5 Screen display during the running of the "LUTTER" model

6.4. Example 3.

This example presents a simulation model of the main warehouse of a major computer manufacturer and supplier which was built, using the AWARD system, as part of the work of a Cranfield MSc thesis in logistics and distribution [Jamani 1989]. The model was developed to test the AWARD system and to evaluate its functionality in order to model the company's warehouses and suggest improvements to the system.

6.4.1. Warehouse model description.

Examples 3 and 4, described in sections 6.4 and 6.5, were undertaken using a version of the AWARD software which was a significant development of that described in this thesis, and which did not employ the SIMVIS software. This additional software development was undertaken by the AWARD consortium, and the author was not directly involved in this. Within these additional features is the ability to simulate the work (administrative work, labelling, repackaging, etc.) done at reception before the pallets are stored or at despatch before the pallets are loaded into the truck. The time taken to do this work is modelled by putting the pallets in a special type of racking (marshalling areas) where they stay as long as necessary. Figure 6.6 shows the warehouse layout where can be seen the marshalling areas as the racking on the shadow areas, corresponding to the reception (on the right) and to the despatch (on the left).

The warehouse handling equipment (see Fig 6.6) consists of six powered pallet trucks, two swivel-reach trucks, two order-picker trucks, one reach truck, one c/balance truck and one combi truck with different operational characteristics (see [Jamani 1989]). There are four reception bays and five despatch bays. The racking is five levels high APR/SD (Adjustable Pallet Racking/Single Deep). The racking module has the following dimensions: width = 1.452m; depth = 1,15m; height - 1.525m.

The warehouse operates with two shifts (9:00 to 17:30 and 15:00 to 23:30) with different transporters and bays availability (see [Jamani 1989]). The 1500 part numbers were grouped in ten groups with different characteristics. For a more detailed description of the model data see [Jamani 1989].

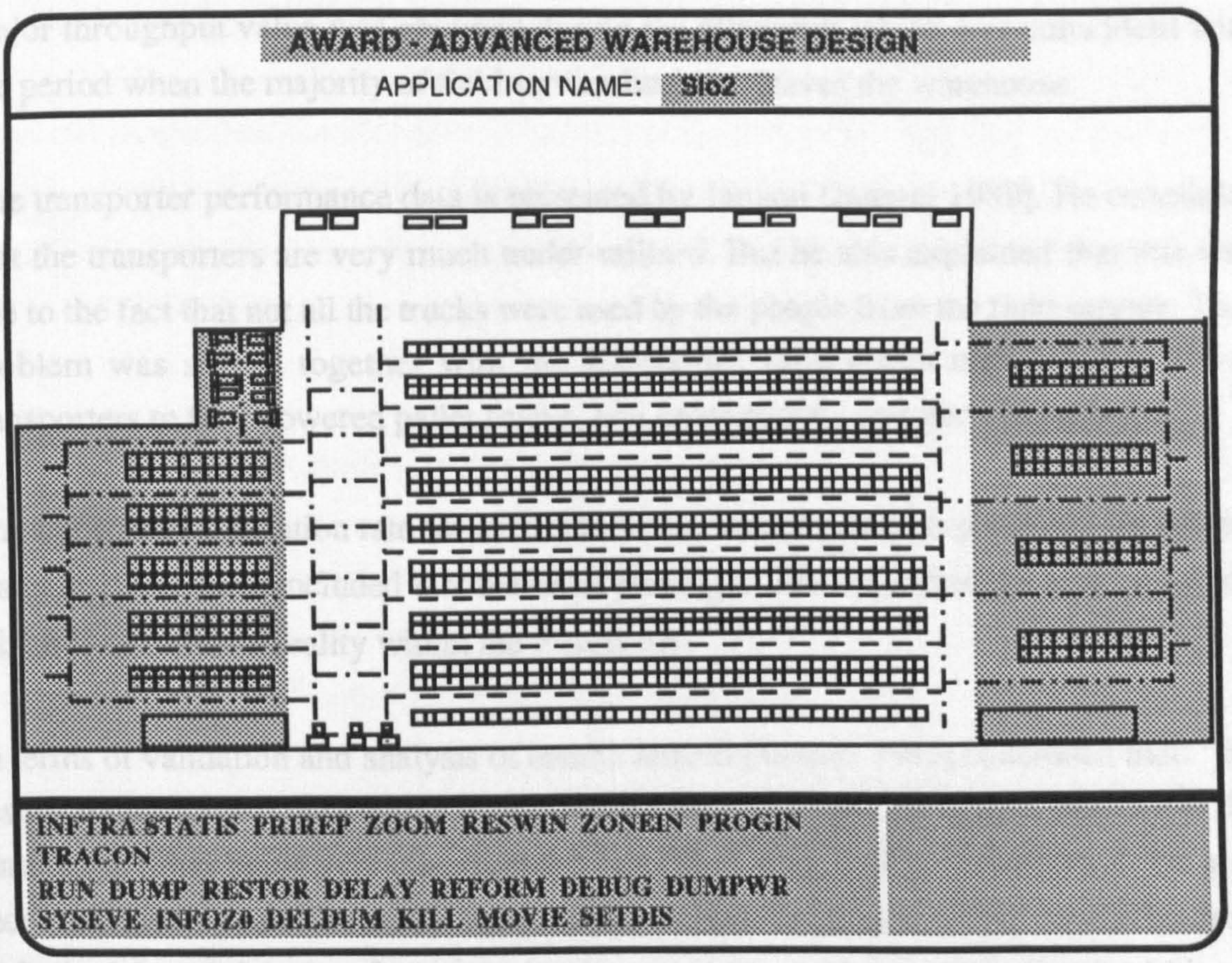


Figure 6.6 Warehouse model (draft) developed as part of a MSc thesis work [Jamani 1989]

6.4.2. Conclusions.

The model developer [Jamani 1989] state that he had not encountered any major problems during the data capture phase. However, he refers that the grouping of the products was time consuming as the company did not provide any grouping of the different part numbers.

The model was run during a typical warehouse working day and a report was generated every sixty minutes. The simulation results analysis were concentrated on the following items: throughput of warehouse; transporter performance data; and transporter utilisation.

The case and pallets throughput obtained by running the simulation model was compared with the real warehouse throughput and similar paths were observed. The

major throughput value was obtained during the afternoon which was coincident with the period when the majority of field service business leaves the warehouse.

The transporter performance data is presented by Jamani [Jamani 1989]. He concludes that the transporters are very much under-utilised. But he also explained that this was due to the fact that not all the trucks were used by the people from the field service. This problem was solved together with the warehouse supervisors reducing the initial transporters to four powered pallet trucks, two order pickers and one swivel truck.

The transporter utilisation rate for each warehouse working hour is presented by Jamani [Jamani 1989]. He concluded that the order picking trucks were used the most regularly which is also true in reality within the warehouse.

In terms of validation and analysis of results Jamani [Jamani 1989] concluded that: "the results indicated that the initial data capture for the model was fairly accurate, due to the simulation times being very close to the actual process times of the warehouse". He also recommended that further study measurements of the transporter speeds should be done as he had found the manufacturers speeds were very optimistic. This fact lead him to reduce all the speeds in the model by 33%.

6.5. Example 4.

This example presents the use of AWARD system in a consultancy work done by Cooper, Saw and M.P. Clarke [Cooper & Saw 1990] to evaluate a warehouse proposed configuration. The warehouse was part of the future expansion of the Coca Cola and Schweppes Beverages Ltd soft drinks factory at Wakefield.

The proposed warehouse and handling equipment were fully computerised, and should be capable of controlling up to 600 pallet movement per hour and up to 300 vehicle movements per day. The need for a simulation model resulted from the fact that the static analysis of cycle times, which indicated the need for 27 fork lift trucks, did not allow to analyse the impact of fork lift trucks interacting such as task priorities and potential congestion [Cooper & Saw 1990].

Cooper & Saw justify the use of simulation saying [Cooper & Saw 1990]: "In order to prove that the proposed warehouse configuration and system logic could handle the potential congestion, the changing priorities, and potentially conflicting objectives (and do this through the peaks and throughs of the operation) a simulation model of the warehouse was needed, which was able to reflect the changing conditions in the warehouse over time".

6.5.1. Warehouse model description.

This model was developed using an AWARD version, as in example 3, which handled marshalling areas. The warehouse storage includes different racking types such as block stacking, drive-in racking, Adjustable Pallet Racking (APR) and live storage. There were a total of 53 products.

The pallets arrived from five production lines with a defined hourly output rates. The outorders were generated with reference to other factories. An initial stockholding of 19000 pallets was created based on the proposed stock cover levels for each product. The model was prepared to simulate one weeks warehouse operation. For a more detailed description of the model data see [Cooper & Saw 1990].

6.5.2. Conclusions.

The use of the simulation model by people working directly in warehouses is a very valuable experience. The direct observation of the graphics simulation aids in understanding the model and explains why certain results are obtained. A good example of that is [Cooper & Saw 1990] the causes of fork lift trucks congestion and resultant queuing time.

Another important aspect is the "what if" analysis which is encouraged by the visual interactive simulation system. Cooper & Saw indicate the following areas where they have introduced changes in the model [Cooper & Saw 1990]:

- allocation priorities of tasks to fork lift trucks;
- numbers of fork lift trucks for preferred tasks;
- introduction of one way traffic flow in some aisles;

- increased pre-assembly lead time for goods outwards;
- adjustment to fork lift trucks travel speeds to allow for acceleration and deceleration due to the number of turns.

About the way they introduce changes in the model Cooper & Saw say that in one simulation run, where the results were not as expected, they were not able to take any conclusions because the many changes made did not allowed them to identify the implications of each change. So they have restricted the number of changes at a time.

The use of the simulation model has helped to achieved confirmation for [Cooper & Saw 1990]:

- the suitability of the warehouse layout;
- the allocation of preferred products by zone;
- the ability of the operation to handle peak work loads;
- numbers and types of fork lift trucks;
- vehicle loading schedules;
- task priorities;
- task allocation to fork lift trucks by number and type.

Finally Cooper & Saw conclude that the simulation model provides invaluable information in the design of a warehouse. However they draw our attention to the fact that the data preparation and quality is crucial for obtaining good results. They highlight the importance of the model logic to interpret the proposed operation, which requires from the modeller a good understanding of the problem and from the user the confirmation that the model logic representation has the acceptable accuracy. They also state that the visual graphics simulation increases the confidence in the model.

6.6. AWARD Enhancements.

The AWARD system, since the version (IA) described here, has been enhanced in several aspects. The major upgrade was called version (IB) and was developed by J.B. Basto (GEIN/University of OPorto - Portugal) and M. Clarke (DSU/Cranfield - UK). This version incorporated the following areas of functionality:

- pickup and delivery stations;
- marshalling;
- housekeeping;

- multiple pick locations;
- user specified inorders.

Pickup and delivery stations are ancillary racking used to temporarily store pallets. These stations are used as intermediate buffers or to allow changes in the type of handling equipment during the transport of pallets. Usually they are used at the entrance of narrow aisles having dedicated transporters.

Marshalling areas are racking located inside reception and despatch areas. The marshalling concept was introduced to model the work done at reception and despatch (e.g. packing, labelling, quality control, etc.). This is done by putting the pallets in the racking and force them to stay there for a pre-defined period of time.

Housekeeping is the task of rearranging stock by moving pallets to other areas with the objective of improving the throughput during peak times. This was implemented as a new job type, usually with low priority, consisting of moving pallets from one reserve storage zone to another with higher priority and having empty cells.

Multiple picking locations were implemented to allow a product to have cells acting as reserve picking locations. This allows a change of the active picking location when the current one is emptied.

User specified inorders were developed to allow the time of arrival and the inorder data to be specified by the user instead of being automatically generated by the system. A detailed description of the pick up and delivery stations and marshalling implementation can be found in [Basto 1991].

J.B. Basto [Basto 1991] has developed a new package from the software described here to enable the modelling of raw material storage for supplying production systems. The logic incorporated allows the handling equipment to supply several machines with raw material and take it back to storage if necessary. The machines can be in four different states: set up; working; out of stock; and waiting raw material. The use of colour code to represent the machines states allows a very good perception of the system operation performance.

CHAPTER 7

CONCLUSIONS

7. Conclusions

7.1. Achievement of Objectives

In Chapters 2 and 3 simulation concepts have been presented and the different approaches to model building have been reviewed. This review is organised and presented from the practical point of view of someone who wants to develop a simulation model. The detailed description of the simulation systems is intended to show how within each approach a model is built and what skills are required. Particular attention is given to Visual Interactive and user friendly simulation software as it is believed that only these can be used by non-technical people. Hence the first objective, to review the current position of simulation software, is achieved in these two chapters.

In Chapter 4 a description of the warehouse design process is given and common difficulties are highlighted. The warehouse design objective is defined as the selection of the system which provides the most efficient and economic flow of goods at a minimum cost of space, labour and equipment. In fact, the designer's experience is vital in selecting the best feasible solution as many factors are involved and there is a lack of tools to evaluate different configurations .

As warehouses increase in size and complexity, with increasing application of automation, then more powerful tools will be needed to reach a feasible solution meeting the projects timescales. With this in mind a Decision Support System (DSS) framework for warehouse design has been proposed to give support during all the design phases.

From the conceptual description of the DSS it is obvious that the detail design is more important than the outline design in the overall design process. Support during the outline design allows the faster selection of possible configurations, but computer-aided detail design enables the definition and evaluation of each configuration to check if it performs according to specification. This explains why the DSS detail design stage has been chosen to be developed further.

For the evaluation of different warehouse configurations, simulation assumes a very important role and sometimes appears as the only solution. Several reasons have been given, demonstrating the need for warehouse simulation. The overview of available

solutions for warehouse simulation and their qualitative comparison have shown that, unless the warehouse designer is also a simulation expert, he will not be able to use the current simulation software to build warehouse models. This is due to the fact that the software which has enough power to do the job is too complex to use and the software which is easy to use is inadequate for warehouse modelling.

The examples taken from the literature show that the warehouse simulation models developed are either simplistic or just model some parts of the warehouse. The model developed using PCMODEL shows that the use of a simple simulation system involves many limitations when it comes to building even a simplified warehouse model. The EFACEC model shows the amount of work and complexity involved in building a full warehouse model using visual interactive graphics simulation, and also the level of expertise which is required.

The analysis of how appropriate current simulation software is for warehouse modelling has led to the conclusion that warehouse systems have special characteristics which require a new way of defining the simulation model and new modelling elements to represent the complex logic of a warehouse system.

Chapter 5 describes the background, the evolution and the characteristics of the AWARD system. AWARD is a data-driven generic model whose major innovation is the use of Computer Aided Design (CAD) to define the warehouse simulation model. Previous work, done mainly in manufacturing systems, has used numerical data or an icon based interface to build the simulation model. Scaled design, made possible by the use of CAD, is crucial in modelling warehouses as their performance depends mostly of spatial relationships. The racking dimensions, racking access, transporter paths and every aspect of the warehouse layout are essential to the operational behaviour and so must be defined rigorously.

The first two examples from Chapter 6 have been used to increase confidence in the AWARD system. They have been used also to demonstrate the potential of this approach. The other examples from Chapter 6, taken from the literature, show that the system developed can be used successfully for warehouse design and management.

The major innovations of this system are:

- the shorter period of time required to build the simulation model (1 to 2 weeks not months);
- the user friendly interface, not requiring a high level of computer skills (it can be learned by warehouse designers without computer expertise in just a few days);
- the visual interactive graphics simulation (the display is a scaled down design of the warehouse) is well understood by the warehouse designers and increases their confidence in the model, helps them to do the debugging and tuning of the model, problems arising can be seen and alternative solutions are more easily found and finally the use of interactive commands allow the warehouse designer to quickly evaluate different alternatives;
- the quality of the on-line information and final print-out reports (the use of bar charts and warehouse terminology makes the simulation data readily available in a usable form for warehouse designers).

The development of the AWARD system has been done working closely with designers and managers. This relationship has contributed to the third objective of this work, that is creating the most appropriate simulation environment for warehouse design and study purposes.

7.2. Further Work on AWARD.

The experience gained in using the AWARD system in real life applications has generated ideas for its future development. Two obvious aspects to be improved are the user interface and the hardware platform / software tools. These will allow the user to take advantage of the latest evolution in graphical user interfaces and of the increasing power of micro-computers and workstations. Two potential development areas are the interactive definition of warehouse control rules and the link of the simulation with the warehouse control software.

7.2.1. Use of Graphical User Interfaces (GUIs).

As AWARD is a visual interactive simulation software for use by warehouse designers, there would be gains in efficiency and user friendliness by using a Graphical User

Interface (GUI) and a new hardware platform. These interfaces are now available in microcomputers, such as Microsoft Windows working on MS-DOS IBM PC compatibles, and in graphic workstations such as OSF Motif for Unix. The facilities available within these interfaces for software development (e.g. pull-down menus, multiple windows, graphic toolkit libraries, etc.) together with the increasing CPU power (e.g. risc technology) would improve AWARD performance and flexibility.

The use of an Object Oriented Programming language like C++ would allow users to take advantage of the object nature of graphics and simulation. Warehouse equipment would be defined as objects with a graphic representation. Different transporter classes could be created to describe the variety in warehouse handling equipment. The object data would store the transporter dimensions and performance characteristics and the transporter jobs would be implemented as class methods. The warehouse components would be stored in object libraries that could be reused in future simulation models.

7.2.2. Definition of control rules.

One of the limitations found in the current AWARD version is that if different control rules need to be used, then changes in the source code have to be made. The creation of an interactive language to define the warehouse operational logic at the configuration stage and modify it during the running of the simulation, would increase the flexibility of the system. The warehouse operational logic is mainly determined by the transporter modes of operation, transporter control rules and warehouse layout (product zoning).

The transporter modes of operation define the type of jobs the transporters can do and their priorities. In the current version this is done during the configuration stage by choosing from a range of available (hardcoded) jobs and then sorting them by descending priority. As part of future work in AWARD, the definition of the transporter mode of operation could be done not only during the configuration stage but also during the running of the simulation. An interactive definition language could be created to define new modes of operation using simple instructions (e.g. move to, load pallet, if idle then, etc.). Another way of doing this definition could be the use of a transporter learning mode option that would allow the use of the interactive graphics facility to specify the new task.

The transporter control rules define the way decisions are made during the transporter operation. These rules handle situations like transporter congestion, aisle blocking, seizing narrow aisles, assigning transporters to orders and many others. Currently, some of these rules are hardcoded and cannot be changed without changing the source code. The same principles suggested for the definition or modification of the transporter modes of operation could be used here.

The warehouse layout (product zoning) defines the areas where the products are stored and specifies the access logic for storing and retrieving the products. The warehouse layout (product zoning) is made during the configuration stage and includes the creation of the zones and the specification of the cell priorities within each zone. As future developments, the warehouse layout (product zoning) definition could also be done during the running of the simulation. The interactive definition of the racking zones is adequate as it is made now. However, the definition of the cell priorities, associated with the racking creation sequence, should be made more flexible. This could be done by allowing the specification of the priority of each individual cell within the racking zone.

7.2.3. Use of AWARD for evaluation of warehouse control logic.

In many warehouse applications today all or part of the system is driven by control logic held in dedicated computers. The development and implementation of this control software is often a key activity in the warehousing project, indeed, there are many cases where this has been on the critical path. Simulation would seem to have an important application in the evaluation of the logic and the software (see figure 7.1).

To link the simulation model with the control software, running on a separate machine, would allow the testing of the logic before the warehouse was commissioned. If a certain situation needed to be analysed, such as a large outorder arrival, the simulation could be used to study the implications, and to test whether the control software had the required functionality and performance.

Moreover it is often necessary to introduce modifications in the control software when the warehouse is already operating. This fact can be due to changes in the order profile, improving based on practical experience, buying new equipment, reorganising the

layout, etc. The simulation can be used (see figure 7.1) to test and evaluate the software modifications before they are implemented. This is very important as testing the software modifications directly in the warehouse can lead to high costs, due to the disturbance of the normal work, or to unsafe situations caused by software malfunction.

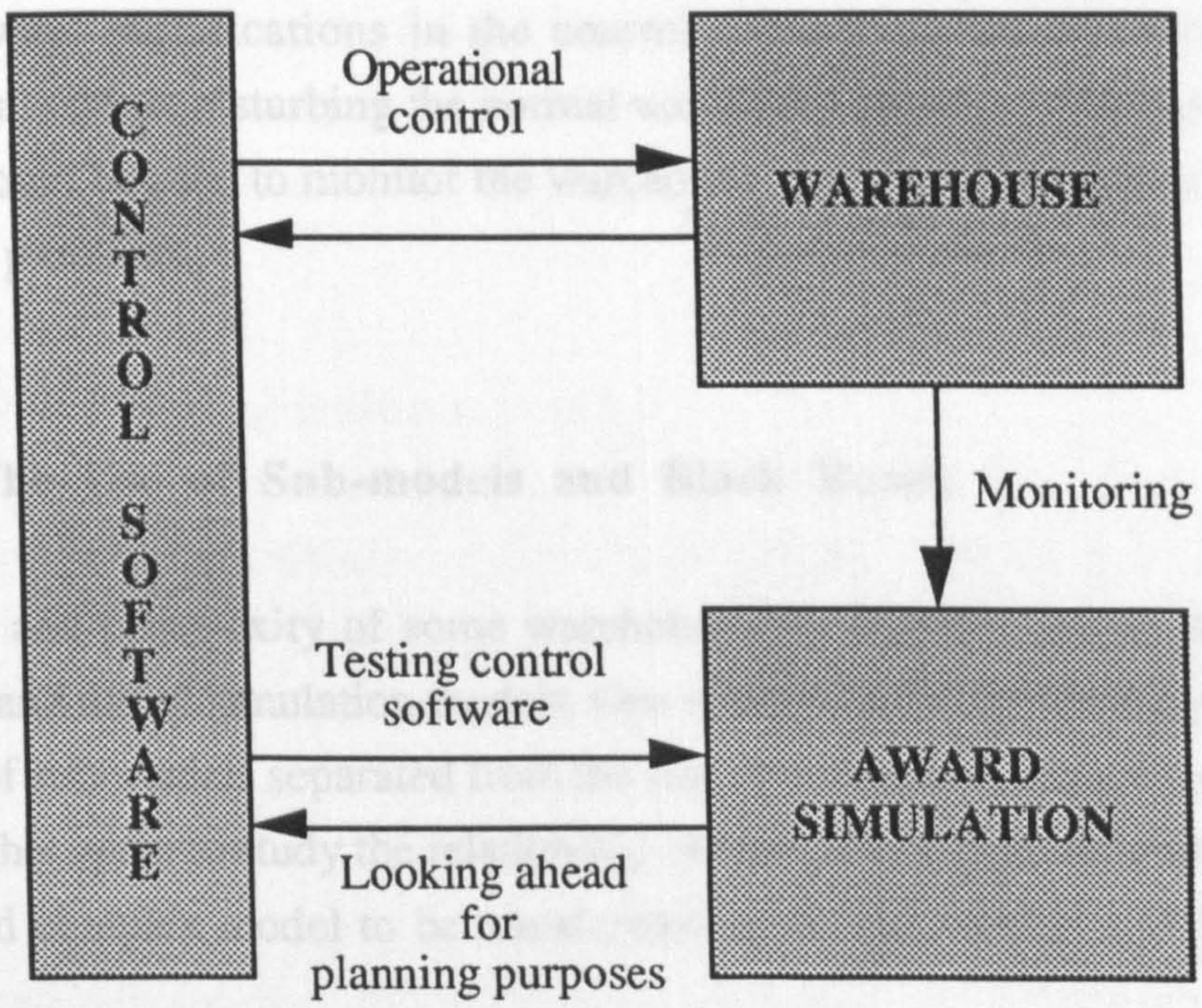


Figure 7.1 AWARD as a warehouse evaluation and planning tool

7.2.4. The Use of AWARD as an operational planning tool.

Existing use of AWARD has concentrated on the design process and the evaluation of strategic alternatives. If a simulation model is maintained in an operational environment and the logic is close to the actual control logic then this model can be used on a day to day basis for the evaluation of short term decisions. This, of course, means that the model has the ability to read order files from the operational system on a regular basis with no manual effort.

The definition of a standard interface between the control software and the simulation would allow the simulation to use the control software logic instead of its own. This

would have the advantages of using exactly the same logic as used in the warehouse control software and the simulation would be free from having to implement the warehouse control logic.

The use of this interface would offer several benefits (see figure 7.1). During the warehouse operation the simulation could be used to look ahead for planning purposes and the results could then be fed directly into the control software and used straightaway. Modifications in the control software could be evaluated using the simulation without disturbing the normal warehouse operation. The simulation graphic display could be used to monitor the warehouse operations, helping in supervising and detecting problems.

7.2.5. The Use of Sub-models and Black Boxes.

The size and complexity of some warehousing systems can cause difficulties in the creation and use of simulation models. One solution to overcome this problem could be the use of sub-models separated from the rest of the system, analysed in detail and then put together again to study the relationship between them. This approach would enable a large and complex model to be transformed in several smaller and more simple sub-models.

Another solution could be the use of certain areas simulated as black boxes. The interaction between the black boxes and the model could be defined as a flow of products. The flow of products could be modelled as outorders arriving at regular intervals. The use of this approach could be used to model production systems with warehousing facilities.

7.2.6. Further Development of the DDS for Warehouse Design.

AWARD current version implements the simulation module from the Decision Support System (DSS) Framework described in chapter 4. This corresponds to the two boxes "design using CAD techniques" and "visual interactive simulation" from Fig. 4.1. As it is described in chapter 4 the use of expert systems, technological data bases, cost and

performance functions could be used to develop further AWARD to give user support during all the warehouse design stages.

Introducing expert systems in the outline design phase and in the analysis of the simulation results should be the next DSS research development. Warehouse design is based mainly on the designer experience. Many decisions can be made by looking to the technological solutions adopted in similar cases. A knowledge data base with existing solutions and a rules data base reflecting the experience of designers are essential for warehouse design support. Both data bases should be continuously updated to reflect new technological solutions and to capture the knowledge from warehouse designers experts.

With AWARD current version it is already possible to build up and run a warehouse simulation model in a short period of time and without being a simulation expert. However interpreting simulation results and evaluating different configurations still require hard work and experience. The use of an expert system would help to analyse the simulation results and to decide, from previous experience, if one configuration would be better than another.

7.3. Implications for simulation software.

7.3.1. Implications of AWARD in the simulation evolution.

In the past few years software developers have been concerned with the development of easy to use software with user friendly interfaces. This is due mainly to the fact that each day more people with less computer expertise make use of computers. Simulation software has also followed this trend. Figure 7.2 shows the AWARD implications in the simulation evolution by using a CAD driven generic model for warehousing systems simulation. The use of CAD has been introduced because the use of data and icon based definition of simulation models, commonly used in manufacturing systems, was inadequate for the complete definition of warehouse models.

The data and icon based simulation model definition allows a pictorial representation of the system with a level of detail that is good enough for most of the manufacturing systems. In manufacturing systems the key factors are the process plans, the utilisation

of the processing machines and the flow of parts through the system. As machining is the prime activity in a production unit the machine layout and the movement of the parts are often of minor importance in the system performance. The focus is on evaluating different scheduling rules using efficiency measures such as machine utilisation or the time taken to finish the jobs. So it is common practice to use average values for transporter travel times and define their path as a series of pickup and dropoff points.

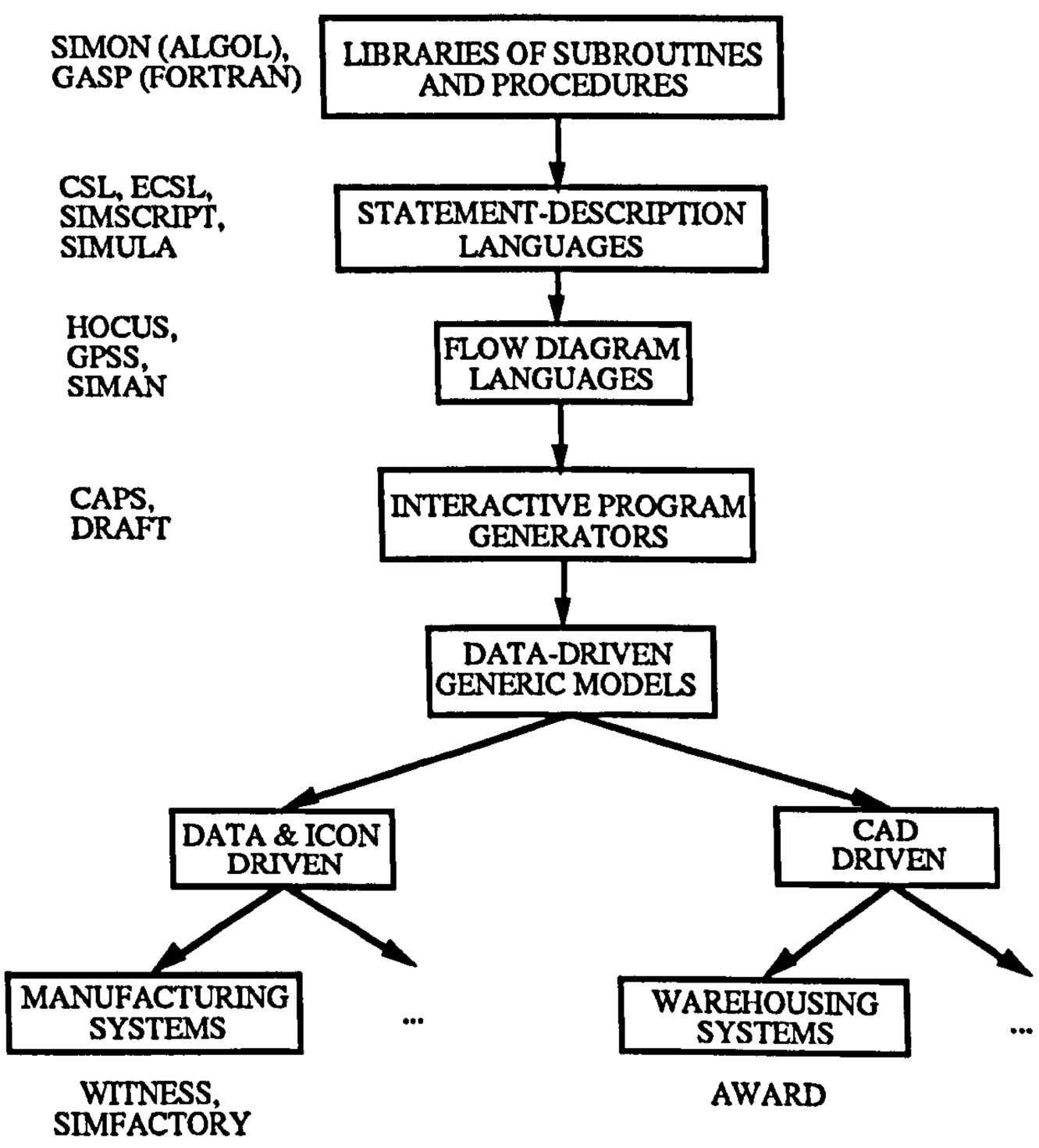


Figure 7.2 Implications of AWARD in the simulation evolution

Manufacturing simulation systems are inadequate for warehousing simulation because movement is the prime activity in a warehouse ant it can not be realistic represented using an icon based model definition. In warehousing systems the key factors are racking layout, racking storage type and transporters. The focus is on evaluating different racking layout and on the selection of the number and type of transporters using efficiency measures such as the time taken to satisfy orders or product throughput. A different approach from the manufacturing systems is required for the simulation model definition in order to capture the necessary data to describe the spatial activity in a warehouse. CAD has been used as a new approach to define warehouse simulation models.

To describe warehouse spatial activity scaled design must be used. Physical locations and travel distances must be entered accurately. It would be time consuming and error prone to enter all this data numerically. Computer Aided Design has the ability to allow an easy and accurate way of defining the warehouse layout. Furthermore there are many advantages in associating graphic objects representing warehouse components such as racking or transporters with the simulation elements used to represent them in the simulation model. The use of in-built CAD system in simulation software speeds up the process of developing the model and also allows its definition using an object oriented approach. This is an advantage against the use of an existing CAD system. The in-built CAD system solution is better than the use of existing CAD systems because it is less complex as it has only the necessary features to design the warehouse, it is less expensive than most CAD packages, it does not require long learning periods and also it can incorporate a technical terminology that is known by warehouse designers.

The approach used in this work allows warehouse designers to have easier access to the use of simulation. The graphic animation and visual interactive techniques help the analyst to understand the system, to identify problems and to search for possible solutions. Nevertheless the analyst must be aware that he is using a simulation model and should act accordingly. The simulation model should not be taken as the reality and the analyst must have enough simulation expertise to be able to draw credible conclusions.

7.3.2. Applications where the AWARD approach is appropriate.

The AWARD approach is appropriate whenever the spatial factor has a predominant influence in the behaviour of the system to be modelled. If the physical location of the objects and the path of moving parts are crucial for the study of the system then they must be defined precisely. The use of CAD enables the accurate definition of the objects dimensions, objects position, travel paths and distances. This approach could be used to develop simulation systems in many different areas such as: road systems; super-markets; airports; railway systems; factory layout; container parks; and air traffic control. In road systems the network could be defined in a similar way to the one used in AWARD but allowing the use of curves. An option to input data from a map using a scanner should be included. The characteristics of each road (e.g. capacity, average speed, etc.) could be defined using numerical data. The traffic could be represented as

graphic dots where queues forming and junction congestion could be easily seen on the screen. Particular attention should be given to modelling traffic movement.

In super-markets the racking layout could be defined as in AWARD. People could be represented as graphic dots. Special attention should be given to modelling people movement and to the definition of shopping lists. Observing people congestion on the simulation display could help to modify the goods layout or to determine the number of cash registers. Railway systems are similar to road systems. However train movement is more easily modelled. The railway network could be defined as in road systems. The system could be used to evaluate time tables or to study train scheduling in railway stations. Factory layout and container parks are very similar, in the way the layout is defined, with warehousing. In factory layout new simulation elements should be created to handle production machines. Container parks could be treated as block stacking in AWARD but new logic should be included to handle the containers way out sequence. These systems would benefit as AWARD of the visual detection of problems.

7.3.3. Applications where CAD approach is not appropriate.

The CAD approach is not appropriate when the layout and distances between objects are not important for the study of the system to be modelled. Good examples can be found in manufacturing systems where average values can be used for the time taken to move the parts from one machine to another and the machines layout is usually irrelevant in the system performance.

7.4. Simulation in the warehouse design process.

Simulation has a very important role in the warehouse design process. The warehouse large size, great complexity, high level automation and high investments are factors that make warehouse design a complex task. The lack of tools to help designers during the project development make it difficult to obtain feasible solutions. The success of the project depends mainly on the experience of the design team. Decision Support Systems with the objective of helping warehouse designers to achieve better solutions are extremely important today. Simulation is the best, and probably the only, method available to analyse the complex relationships between all the components of a modern warehouse. The AWARD system allowing the definition of simulation models in short periods of time and by using an user friendly interface help designers to improve the design and operation of warehouses.

REFERENCES

Alemparte, M., Chheda, D., Seeley, D. & Walker, W., "Interacting with discrete simulation using on-line graphic animation", Comput. Graph., 1, 309-318, 1975.

Amiry, A.P., "The simulation of information flow in a steelmaking plant", Digital Simulation in Operational Research (Hollingdale ed.), 347-356, English Universities Press, London, England, 1965.

Balci, O., "Credibility Assessment of Simulation Results", Proceedings of the 1986 Winter Simulation Conference (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Balci, O. & Nance, R. E., "Simulation Model Development Environments: A Research Prototype", J. Opl. Res. Soc., 38, 753-763, 1987.

Balmer, D. W., "Modelling Styles and Their Support in the CASM Environment", Proceedings of the 1987 Winter Simulation Conference (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Balmer, D. W. & Paul, R. J., "CASM - The Right Environment for Simulation", J. Opl. Res. Soc., 37, 443-452, 1986.

Basto, J. A. B., "Novos Desenvolvimentos de um Gerador de Modelos de Simulação Visual Interactiva para Concepção de Armazéns", Trabalho de Síntese elaborado para prestação de Provas de Aptidão Pedagógica e Capacidade Científica, FEUP, Oporto University, 1991.

Bastos, J. M., "SIMVIS - Simulação Visual e Interactiva", Trabalho de Síntese elaborado para prestação de Provas de Aptidão Pedagógica e Capacidade Científica, FEUP, Oporto University, 1985.

Bastos, J. M. & Moreira da Silva, C., "SIMVIS: A Visual Interactive General Purpose Simulation System", Progress Report Nº 2, GEIN - Department of Mechanical Engineering of Oporto University, 1985.

Bell, P. C., "Stochastic Visual Interactive Simulation Models", J. Opl. Res. Soc., 40, 615-624, 1989.

Bell, P. C., "Visual Interactive Modelling in Operational Research: Successes and Opportunities", J. Opl. Res. Soc., 36, 975-982, 1985.

Bell, P. C. & O'Keefe, R. M., "Visual Interactive Simulation - History, Recent Developments, and Major Issues", SIMULATION, 49:3, 109-116, 1987.

Bozer, Y. A., Branigan, M. J. & Mullens M. A., "Design Warehousing Systems", Presented at TIMS/ORSA Detroit Joint National Meeting, April 19, 1982.

Branigan, M. J., "Visual Interactive Simulation for Automated Warehouse Design", Proceedings of the 9th International Conference 'Automation in Warehousing', Brussels, Belgium, 1988.

Brito, A. C., "MSC - A Data-entry Screen Generator", Internal Report, Computer Center of the Mechanical Engineering Department of the Oporto University - Portugal, 1985.

Brito, A. C., "FORTRAN 77 graphics library for tektronix 4109/4107 terminals", Internal Report, Computer Center of the Mechanical Engineering Department of the Oporto University - Portugal, 1984.

Brito, A. C. & Bastos, J. M., "Input/Output subroutines library", Internal Report, Computer Center of the Mechanical Engineering Department of the Oporto University - Portugal, 1985.

Brown, J. C., "Visual Interactive Simulation: further developments towards a generalized system and its use in three problem areas associated with manufacturing", M.Sc. thesis, University of Warwick, School of Industrial and Business Studies, England, 1978.

Buxton, J. N., "Simulation Programming Languages", Proceedings of the IFIP Conference on Simulation Programming Languages, North-Holland Publishing Company - Amsterdam, 1967.

Buxton, J. N. & Laski, J. G., "Control and Simulation Language", The Computer Journal, 5, 3, 1962.

CACI Products Company, "SIMSCRIPT II.5 Reference Handbook", , 1989.

CACI Products Company, "PC SIMSCRIPT II.5 Introduction and User's Manual", , 1988a.

CACI Products Company, "SIMGRAPHICS User's Guide and Casebook", , 1988b.

CACI Products Company, "SIMFACTORY II.5 Reference Manual", , 1990.

Carrie, A., "Simulation of Manufacturing Systems", John Wiley & Sons Ltd., 1988.

Chew, S. T., "Program Generators for Discrete-Event Digital Simulation Modelling", Ph.D. Thesis, University of London, 1986.

Clementson, A. T., "Extended Control and Simulation Language", Cle Com Ltd, Birmingham, U.K., 1982.

Clementson, A. T., "Simulating With Activities Using C.A.P.S./E.C.S.L. (The British Approach to Discrete-Event Simulation)", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Cobbin, P., "A Tutorial on the SIMPLE_1 Simulation Environment", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Cobbin, P., "The SIMPLE_1 Simulation Environment", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Conway, R. & Maxwell, W., "Modeling Asynchronous materials Handling in XCELL+", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Cooper, S. & Saw, R., "Computer Simulation of a Large Soft Drinks Warehouse", 'Warehouse & Distribution Software Conference - Technology for profit', Volume one, 13-14, London, February, 1990.

Cox, S., "Interactive graphics in GPSS/PC", SIMULATION, 49:3, 117-122, 1987a.

Cox, S. W., "The Interactive Graphics and Animation of GPSS/PC", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. kelton, eds.), 1987b.

Crain, R. C., Brunner, D. T. & Henriksen, J. O., "Advanced Features of GPSS/H", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. kelton, eds.), 1987.

Crookes, J. G., "Simulation in 1981", Eur. J. Op. Res., 9, 1-7, 1982.

Crookes, J. G., "Generators, Generic Models and Methology", J. Opl. Res. Soc., 38, 765-768, 1987.

Crookes, J. G. & Valentine, B., "Simulation in Micro-Computers", J. Opl. Res. Soc., 33, 855-858, 1982.

Crookes, J. G., Balmer, D. W., Chew, S. T. & Paul, R. J., "A Three-Phase Simulation System Written in Pascal", J. Opl. Res. Soc., 37, 603-618, 1986.

Donovan, J. J., Jones, M. M. & Alsop, J. W., "A graphical facility for an interactive simulation system", Proceedings IFIP Congress 1968, Amsterdam, North-Holland, 1969.

Doukidis, G. I. & Paul, R. J., "Research into Expert System to Aid Simulation Model Formulation", J. Opl Res. Soc., 36, 319-325, 1985.

DSU, GEIN & C&L, "Advanced Warehouse Design Workshop", Distribution Studies Unit - Cranfield, GEIN - University of Oporto, Coopers and Lybrand Associates, 15th January, 1987.

Egarr, S., "Getting it Right - A Layman's Guide to Warehouse System Design", Factory Equipment News, pp 20-23, January, 1984.

Ellison, D., Herschdorfer, I. & Tunnicliffe-Wilson, J., "Interactive simulation on a microcomputer", SIMULATION, 38:5, 161-175, 1982.

Fiddy, E., Bright, J. G. & Hurrion, R. D., "SEE-WHY: Interactive Simulation on the Screen", Proceedings of the Institute of Mechanical Engineers, C293/81, 167-172, 1981.

Firth, K., "Materials Handling System Design and Warehouse Operation", in 'Handbook of Physical Distribution Management', 3rd. edition, J. Gattorna (ed.), Gower Pub. Comp. Ltd., 1983.

Fisher, M. W. J., "The application of visual interactive simulation in the management of continuous process chemical plants", PhD. thesis, University of Warwick, 1982.

Fishman, G. S., "Concepts and Methods of Discrete Event Digital Simulation", Wiley, New York, 1973.

Fishman, G. S., "Principles of Discrete Event Simulation", John Wiley & Sons, 1978.

Garzia, R., "Simulating With GPSS/PC", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Gilman, A. R. & Watremez, R. M., "A tutorial on SEE WHY and WITNESS", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Gordon, G., "A General Purpose System Simulation Program", Proc. EJCC, Washington, D.C., pp.87-104, New York, Macmillan Publishing Co., Inc., 1961.

Gordon, G., "System Simulation", Prentice-Hall Inc., 1978.

Gordon, G., "The Design of the GPSS Language", in Adam, N. R. & Dogramaci, A., 'Current Issues in Computer Simulation',Academic Press, New York, 1979.

Grant, J. W. & Weiner, S. A., "Simulation Series, Part 4: Factors To Consider In Choosing A Graphically Animated Simulation System", Industrial Engineering, August, 1986.

Greenberg, S., "GPSS Primer", Wiley, New York, 1972.

Guimarães, R. C., Moreira da Silva, C., Brito, A. C., "SIMAHID: Sistema de Simulação Visual Interactiva da Exploração de Aproveitamentos Hidroeléctricos em Regime de Cheias", Internal Reports Nº1 and Nº2, GEIN - Department of Mechanical Engineering of Oporto University, 1985.

Haider, S. W. & Banks, J., "Simulation Series, Part 3: Simulation Software Products For Analyzing Manufacturing Systems", Industrial Engineering, July, 1986.

Harverty, J. P., "Grail GPSS: graphic on-line modelling", Rand Corp, Santa Monica, CA, 1968.

Healy, K. J., "Cinema Tutorial", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Hills, P. R., "An Introduction to Simulation Using SIMULA", NCC Publication 5-Ss, Norwegian Computing Centre, Oslo, 1973.

Hills, P. R., "SIMON - a Simulation Language in ALGOL", in 'Simulation in Operational Research', Ed. S.M. Hollingdale, English Universities Press, London, 1965.

Hitchens, M., "Real Time Artificial intelligence and Simulation in Warehousing", Proceedings of the AutoCADCON conference, Volume 1, CADCON Corp., 1987.

Hollocks, B., "Simulation and the Micro", J. Opl. Res. Soc., 34, 331-343, 1983a.

Hollocks, B., "FORSSIGHT is better than hindsight-visual interactive simulation for plant design, development, and operations", 'IEE colloquium on computer aided engineering in design', IEE, London, England, 1983b.

Hurrion, R. D., "The Design, Use and Requirements of an Interactive Visual Computer Simulation Language to Explore Production Planning Problems", Ph.D. thesis, University of London, 1976.

Hurrion, R. D., "An investigation of visual interactive simulation methods using the job-shop scheduling problem", J. Opl. Res. Soc., 29, 1085-1093, 1978.

Hurrion, R. D., "Visual interactive (computer) solutions for the travelling salesman problem", J. Opl. Res. Soc., 31, 537-539, 1980.

Hurrion, R. D., "Visual interactive modelling", Eur. J. Op. Res., 23, 281-287, 1986.

Hurrion, R. D., "Graphics and Interaction", in 'Computer Modelling for Discrete Simulation' (M. Pidd, ed.), John Wiley & Sons Ltd., 1989.

Insight International Limited, "OPTIK Software Description", The Quadrangle, Woodstock, Oxford, England, 1983.

ISTEL, "WITNESS User Manual", Beachwood, Ohio, 1987.

Jamani, S., "An Evaluation of the Application of Visually Interactive Simulation to the Logistics Function of a Large International Eçectronocs Manufacturer and Supplier", M.Sc. Thesis, Distribution Studies Unit, Cranfield Institute of Technologiy, 1989.

Jasany, L. C., "Automate first with Simulation Software", Production Engineering, Vol.33, No.11, 40-41, 44-46, 1986.

Kachitvichyanukul, V., (Chair), "Simulation Environment of the 1990's (Panel)", Proceedings of the 1987 Winter Simulation Conference (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Katske, J., "User's guide NGPSS: superset of GPSS V for batch and interactive version", Norden report 4059, R0001, Norwalk, CT 134, 1975.

Keith, A. & Porter, K., "A Case Study of the Application of Computer Simulation to Automated Storage and Handling Systems", Proc. of the Inst. of Mech. Eng. International Conference on Advanced Handling Systems - Applications and Experiences, London, 1988

Kilgore, R. A. & Healy, K. J., "Animation Design with Cinema", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Kirkpatrick, P. & Bell P. C., "Visual Interactive Modelling in Industry: Results from a Survey of Visual Interactive Model Builders", INTERFACES, Vol.19, No.5, pp.71-79, 1989.

Kiviat, P. J., Markowitz, H. M. & Villanueva, R., "SIMSCRIPT II.5 Programming Language", CACI, Inc., Edt. by E. C. Russel, Los Angeles, California, 1987.

Kiviat, P. J., Villanueva, R. & Markowitz, H. M., "SIMSCRIPT II.5 Programming Language", CACI inc., Los Angeles, 1973.

Kleine, B., "A Simfactory Tutorial", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Law, A. M. & Haider, S. W., "Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey", Industrial Engineering, May, 1989.

Law, A. M. & Kelton, W. D., "Simulation modeling and analysis", McGraw Hill, 1982.

Macintosh, J. B., Hawkins, R. W. & Sheppard, C. J., "Simulation on microcomputers - the development of a visual interactive modelling philosophy", Proceedings of the 1984 Winter Simulation Conference, (Sheppard, Pooch and Pedgen, eds.), 1984.

Markowitz, H. M., Hausner, B. & Kan, H. W., "SIMSCRIPT: A Simulation Programming Language", RAND Corporation, RM-3310-pr., Prentice-Hall, New Jersey, 1963.

Mathewson, S. C., "Simulation program generators", SIMULATION, 23:6, 181-189, 1974.

Mathewson, S. C., "A Programming Manual for SIMON Simulation in FORTRAN", Imperial College, London, 1977.

Mathewson, S. C., "Simulation Program Generators: Code and Animation on a P.C.", J. Opl. Res. Soc., 36, 583-589, 1985.

Mathewson, S. C., "Simulation Support Environments", in 'Computer Modelling for Discrete Simulation', edited by Michael Pidd, John Wiley & Sons, 1989.

Mills, R. I., "A Review of Simulation Languages for Manufacturing Systems Applications", in 'Control and Programming in Advanced Manufacturing', edited by K. Rathmill, IFS (Publications) Ltd, UK, 1988.

Miner, J. & Rolston, L., "MAP/1 User's Manual", Pritsker & Associates, Inc., 1986.

Minnee, H., "Optimisation of a Steel Plate Warehouse with Simulation", Proc. 4th Int. Conf. 'Simulation in Manufacturing', 3-8, 1988.

Minuteman Software, "GPSS/PC General Purpose Simulation Reference Manual", , 1986.

Moreira da Silva, C. & Bastos, J. M., "VISUALPLAN: A Visual Interactive Simulation System for FMS Modelling", Progress report Nº 3, GEIN - Department of Mechanical Engineering of Oporto University, 1984.

Moreira da Silva, C. A. R., "The development of a decision support system generator via action research", Ph.D. thesis, University of Warwick, 1982.

Nance, R. E. & Arthur, J. D., "The Methodology roles in the Realization of a Model Development Environment", Proceedings of the 1988 Winter Simulation Conference (M. Abrams, P. Haigh and J. Comfort, eds.), 1988.

Neelamkavil, F., "Computer Simulation and Modelling", John Wiley & Sons, 1987.

O'Keefe, R. M., "What is Visual Interactive Simulation ? (and is there a methodology for doing it right ?)", Proceedings of the 1987 Winter Simulation Conference, (Thesen, Grant and Kelton , eds.), 1987.

O'Keefe, Robert M., "The three-phase approach: A comment on 'strategy-related characteristics of discrete-event languages and models'", Simulation 47:5, 208-211, 1986.

O'Reilly, J. J. & Lilegdon, W. R., "SLAM II Tutorial", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Palme, J., "Moving pictures show simulation to user", SIMULATION, 29:6, 204-209, 1977.

Paul, R. J., Chew, S. T., "Simulation Modelling Using an Interactive Simulation Program Generator", J. Opl. Res. Soc., 38, 735-752, 1987.

Paul, R. J., Doukidis, G. I., "Further Developments in the Use of Artificial Intelligence Techniques which Formulate Simulation Problems", J. Opl. Res. Soc., 37, 787-810, 1986.

Pedgen, C. D., "Introduction to SIMAN", System Modeling Corporation, 1984.

Pedgen, C. D., "Introduction to SIMAN", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Pidd, M., "Computer Simulation For Operational Research in 1984", in 'Developments in Operational Research' edited by R. W. Eglese and G. K. Rand, University of Lancaster, UK, Pergamon Press, 1984.

Pidd, M., "Computer Simulation in Management Science 2nd ed.", John Wiley & sons, 1988.

Pidd, M., "A Warehouse Simulation on a Small Personal Computer", IJPD & MM 16, 3, 1986.

Poole, T. G. & Szymankiewicz, J. Z., "Using Simulation to Solve Problems", McGraw-Hill, 1977.

Pritsker, A. A. B., "The GASP IV Simulation Language", Wiley, New York, 1974.

Pulat, P. & Pulat B., "Throughput Analysis in an Automated material Handling System", SIMULATION, MAY, 195-198, 1989.

Raghunath, S., Perry, R. & Cullinane, T., "Interactive Simulation Modelling of Automated Storage Retrieval Systems", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Rietz, F., "Cheaper, Better and Quicker - Warehouse Design Using Visual Interactive Simulation", The Institution of Production Engineers, Seminar on 'Simulation - a Tool for Manufacturing Management', London, 1985.

Rubens, G. T., "A study of the use of Visual Interactive Simulation for decision making in a complex production system", M.Sc. thesis, University of Warwick, School of Industrial and Business Studies, England, 1979.

Russell, E. C., "SIMSCRIPT II.5 Tutorial", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Russell, E. C., "SIMSCRIPT II.5 and SIMANIMATION a Tutorial", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Russell, E. C., "Building Simulation Models with SIMSCRIPT II.5", CACI Products Company, 1989.

Schriber, T. J., "Perspectives on Simulation Using GPSS", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Shannon, R. E., "Present and Future Trends in Simulation", Proceedings of the European Simulation Multiconference, (H. Adelsberger and F. Broeckx, eds.), Vienna, Austria, July 7-10, 1987.

Shannon, Robert E., "System simulation: the art and science.", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

SIMSOFT, PCModel Manual, SIMSOFT, San José, California, USA, 1986.

Sohnle, R. C., Tartar, J. & Sampson, J. R., "Requirements for interactive simulation systems", SIMULATION, 20:5, 145-152, 1973.

Standridge, C., Pritsker, A. & Stein, C., "A Tutorial on TESS: The Extended Simulation Support System", Proceedings of the 1987 Winter Simulation Conference, (A. Thesen, H. Grant and W. Kelton, eds.), 1987.

Standridge, C., Vaughan, D., LaVal, D. & Simpson T., "A Tutorial on TESS: The Extended Simulation System", Proceedings of the 1986 Winter Simulation Conference (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Stenzel, J., "Application of Simulation Within the Material Flow Technique and Logistics", Proceedings of the 9th International Conference 'Automation in Warehousing', Brussels, Belgium, 1988.

Sulonen, R. K., "On-line simulation with computer graphics", On-Line 72, 1972.

Szymankiewicz, J., McDonald, J. & Turner, K., "Solving Business Problems by Simulation", McGraw-Hill, 1988.

Tocher, K. D., "The Art of Simulation", English Universities Press, London, 1963.

Tocher, K. D., "Review of Simulation Languages", Operational Research Quarterly, Vol.16, No.2, 1964.

Todd, P. N., "Visual interactive simulation modelling", IEE Computing and control division colloquium on 'Simulation as an aid to plant design', London, England, 1986.

White, D., "PCModel and PCModel/GAF -- Screen Oriented Modeling", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Withers, S. J. & Hurrion, R. D., "The Interactive Development of Visual Simulation Models", J. Opl. Res. Soc., 33, 973-975, 1982.

Wyman, F. P., "Simulation Modelling: A Guide to Using SIMSCRIPT", John Wiley & Sons, 1970.

Zeigler, B. P., "Theory of Modeling and Simulation", John Wiley, 1976.

BIBLIOGRAPHY

Ashayeri, J., Gelders, L. & Van Wassenhove, L., "A microcomputer-based optimisation model for the design of automated warehouses", Int. J. Prod. Res., Vol.23, No.4, 825-839, 1985.

Banks, J. & Carson II, J. S., "Process-interaction simulation languages", Simulation, 44:5, 225-235, 1985.

Banks, J. & Carson II, J. S., "Introduction to Discrete-Event Simulation", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Bell, P. C., "Visual Interactive Modelling as an Operations Research Technique", INTERFACES, Vol.15, No.4, pp.26-33, 1985.

Bertone, L.,Maina, P.,Roveta, R.,Gambini, A.,Turconi, G., "SIMPA: Parametric Detailed Simulator-Emulator for CIM applications", Proceedings of the 2nd International Conference on Simulation i n Manufacturing, 24-26 June, Chicago, USA, 1986.

Boer, C. R. & Metzler, V., "Economic Evaluation of Advanced Manufacturing by Means of Simulation", Material Flow, 3, 215-224, 1986.

Boling, B. A. & Laymon, M. A., "ModelMaster Factory Modelling System Tutorial", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Bozer, Y. A. & White, J. A., "Travel Time Models for Automated Storage/Retrieval Systems", PDRC 82-06, May, 1982.

Carson, J. S., "Simulation Series, Part 2: Convincing Users Of Model's Validity Is Challenging Aspect Of Modeler's job", Industrial Engineering, June, 1986.

Dangelmeier, W. & Bachers, R., "SIMULAP - A Simulation System For Material Flow and Warehouse Design", Material Flow, 3, 207-214, 1986.

Davies, H. & Davies, R., "A Simulation Model for Planning Services for Renal Patients in Europe", J. Opl. Res. Soc., 38, 693-700, 1987.

Davies, R. & O'Keefe R., "Simulation Modelling with Pascal", Prentice Hall, 1989.

Doukidis, G. I., "An Anthology on the Homology of Simulation with Artificial Intelligence", J. Opl. Res. Soc., 38, 701-712, 1987.

Flitman, A. M. & Hurrion, R. D., "Linking Discrete-Event Simulation Models with Expert Systems", J. Opl. Res. Soc., 38, 723-733, 1987.

Flynn, B. B. & Jacobs, F. R., "A simulation comparison of group technology with traditional job shop manufacturing", Int. J. Prod. Res., vol.24, No.5, 1171-1192, 1986.

Graybeal, W. & Pooch U. W., "Simulation: Principles and Methods", Winthrop Publishers, 1980.

Gupta, R. M., "Flexibility in Layouts: A Simulation Approach", Material Flow, 3, 243-250, 1986.

Haddock, J., "An expert system framework based on a simulation generator", SIMULATION, 48:2, 45-53, 1987.

Hartley, M. G. ed., "Digital Simulation Methods", Peter Peregrinus Ltd., 1975.

Hooper, J. W., "Activity scanning and the three-phase approach", Simulation, 47:5, 210-211, 1986.

Hooper, J. W., "Strategy-related characteristics of discrete-event languages and models", SIMULATION, 46:4, 153-159, 1986.

Kelton, W. D., "Simulation Series, Part 5: Statistical Analysis Methods Enhance Usefulness, Reliability Of Simulation Models", Industrial Engineering, September, 1986.

Kleijnen, J. P. C., "Statistical Techniques in Simulation - Part I", Marcel Dekker, Inc., New York, 1974.

Laperrousaz, P., "Simulation: L'atelier Sort de L'ecran", L'usine Nouvelle, Mars, 1987.

Law, A. M., "Simulation Series, Part 1: Introduction To simulation: A Powerful Tool For Analyzing Complex Manufacturing Systems", Industrial Engineering, May, 1986.

Law, A. M. & McComas, M. G., "Pitfalls To Avoid In The Simulation Of Manufacturing Systems", Industrial Engineering, May, 1989.

McGinley, W. G., "Refining Warehouse Design by Simulation", Proceedings of the 9th International Conference 'Automation in Warehousing', Brussels, Belgium, 1988.

McKee, P. R., Robinson, F. D. & Swan, A. W., "An investigation into Materials Handling System of a Fabrication Works", Operational Research Quarterly, 13, 2, 1962.

Morgan, B. J. T., "Elements of Simulation", Chapman and Hall, 1984.

Moser, J. G., "Integration of artificial intelligence and simulation in a comprehensive decision-support system", SIMULATION, 47:6, 223-229, 1986.

Naylor, T. H., Balintfy, J. L., Burdick, D. S. & Chu, K., "Computer Simulation Techniques", John Wiley & Sons, 1966.

O'Keefe, R. M. & Roach, J. W., "Artificial Intelligence Approaches to Simulation", J. Opl. Res. Soc.,38, 713-722, 1987.

O'Reilly, J. J., "SLAM II, Including a Material handling Extension", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Parks, M. W., "Artificial Intelligence Series, Part 1: Expert Systems: Fill In The Missing Link in Paperless Aircraft Assembly", Industrial Engineering, January, 1987.

Payne, J. A., "Introduction to Simulation - Programming Techniques and Methods of Analysis", McGraw-Hill, 1988.

Pidd, M., "Simulating Automated Food Plants", J. Opl. Res. Soc., 38, 683-692, 1987.

Roberts, S. D., "Modelling and Simulation With Insight", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Roberts, S. D., "Modelling and Simulation With Insight", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Schriber, T. J., "Introduction to GPSS", Proceedings of the 1986 Winter Simulation Conference, (J. Wilson, J. Henriksen and S. Roberts, eds.), 1986.

Sheikh, A. R., Paul, R. J., Harding, A. S. & Balmer, D. W., "A Microcomputer-Based Simulation Study of a Port", J. Opl. Res. Soc., 38, 673-681, 1987.

Smith, J. M., Graves, R. J., Kerbache, L., "QNET: An Open Queueing Network Model For Material Handling System Analysis", Material Flow, 3, 225-242, 1986.

APPENDICES

APPENDIX III.A

A GASP IV subroutine for an end of operation event [Carrie 1988]

```
      SUBROUTINE END OF OPERATION
C     Establish which job and machine are involved in the event.
      JOBNO = ATRIB(3)
      IOPNO = ATRIB(4)
      MACNO = ATRIB(5)
C     Find out if there is a job waiting for machine MACNO
C     Search for an entity in file 2 with its 5th attribute equal to MACNO.
      K = NFIND(MACNO, 5, 2, 5, 0.0)
      IF (K .GT. 0) THEN
C     There is a job waiting for machine MACNO, therefore
C     remove the job from the file 2 and observe its attributes.
         CALL RMOVE(K, 2)
         JOB = ATRIB(3)
         JOPNO = ATRIB(4)
C     Start a new operation on this job and schedule and end of operation event.
C     Place an entry in file 1.
         ATRIB(1) = TNOW + OPTIME(JOB, JOPNO)
         ATRIB(2) = 2
         CALL FILEM(1)
      ELSE
C     There is no job waiting for MACNO, set it to idle.
         BUSY(MACNO) = 0.0
      ENDIF
C     Now determine if more operations are required on JOBNO.
      IF (IOPNO .LT. NOOPS(JOBNO)) THEN
C     No more operations are required, therefore the job leaves.
         CALL JOBLEAVES
      ELSE
C     More operations are required. Set attributes for the next operation.
         IOPNO = IOPNO + 1
         MAC = MCREQD(JOBNO, IOPNO)
         ATRIB(3) = JOBNO
         ATRIB(4) = IOPNO
         ATRIB(5) = MAC
C     Is the machine for its next operation idle or busy?
         IF (BUSY(MAC) .EQ. 0.0) THEN
C     Machine MAC is busy, therefore JOBNO must wait.
C     Place JOBNO into file 2.
            ATRIB(1) = TNOW
            CALL FILEM(2)
         ELSE
C     Machine MAC is idle, start new operation and schedule an end of operation event.
            ATRIB(1) = TNOW + OPTIME(JOBNO, IOPNO)
            ATRIB(2) = 2
            CALL FILEM(1)
C     Set machine MAC to busy
            BUSY(MAC) = 1.0
         ENDIF
      ENDIF
      RETURN
```

APPENDIX III.B

A SEE-WHY subroutine for a customer arrival event [Istel 1987]

```
      SUBROUTINE ARRIVL (NXTL)
C     Customer arrival to a newsagent event subroutine
      IMPLICIT INTEGER*2 (I-N)
      CHARACTER*4 CTYPE(5)
      LOGICAL LFULL
      COMMON /USER/ IQ, IWRLD, ISRVQ, IMIX, IDUM
      DATA CTYPE /'A', 'B', 'C', 'D', 'E'/

      IF (.NOT. LFULL(IQ)) GO TO 5
C     If the queue is full increment the 'lost' count in integer attribute 2 of the
C     dummy entity and go to schedule the next arrival
      CALL IINCAT (IDUM, 2, 1)
      GO TO 10

C     Sample from the user distribution to get the customer type

5     ITYPE = IFIX (SAMPLD(IMIX, 2) + 0.5)

C     Store the customer type as integer attribute 1 of the customer entity

      CALL ISETAT (NXTL, 1, ITYPE)

C     Change the description of the customer entity to that corresponding to the
C     customer type

      CALL CHDESC (NXTL, CTYPE(ITYPE))

C     Change the logical display number of the customer entity

      CALL CHILD (NXTL, ITYPE+2)

C     The customer entity is added to the queue tail and removed from the 'world set'

      CALL DADMAX (NXTL, IQ)
      CALL BEHEAD (IWRLD)

C     Sample from a negative exponential to obtain the time of the next arrival

      ARTM = SNEGX (RATTRB(IDUM, 3), 2)

C     Schedule event 1 (arrival),at the time obtained,for the next entity in the 'world set'

      CALL SCHEDL (1, ARTM, IHEDOF(IWRLD))

C     Output the inter-arrival time to the screen

      CALL HREAL (15, 23, 47, ARTM, 5, 2)
      RETURN
      END
```

APPENDIX III.C

Event subroutine from a SIMVIS based simulation model

```
*****************************************************************************
*                              SUBROUTINE   EVEN19
*****************************************************************************
*       TRANSPORTER IN THE PARK  ( JOB 2 )
*
*       Brito, A.E.C., (CCDEMEC - 870914)

        SUBROUTINE EVEN19

        COMMON /AUXSYS/ IPOSYS

*       Reads the pointer to the JOB ENTITY  (event scheduled entity)

        IPOJOB = IDENTI()

*       Reads the job attributes : Pointer to the transporter IPOTRA

        IPOTRA = READAT ( IPOJOB, 2)

*       Reads the system attributes - pointer to the jobs neutral queue
*                                  - pointer to the jobs process.queue

        IPOJNQ = READAT ( IPOSYS, 16)
        IPOJPQ = READAT ( IPOSYS, 22)

*       Take the job from the processing queue

*       ------------------------------
        CALL TAKENT ( IPOJOB, IPOJPQ)
*       ------------------------------

*       Refresh the job

*       ------------------------------
        CALL DANENP ( IPOJOB, 2, 0.0)
        CALL DANENP ( IPOJOB, 3, 0.0)
        CALL DANENP ( IPOJOB, 4, 0.0)
        CALL DANENP ( IPOJOB, 5, 0.0)
        CALL DANENP ( IPOJOB, 6, 0.0)
        CALL DANENP ( IPOJOB, 7, 0.0)
*       ------------------------------

*       Add the job to the neutral queue

*       ------------------------------
        CALL ADDLAS ( IPOJOB, IPOJNQ)
*       ------------------------------

*       Reads the transporter attribute - 47: Breakdown flag

        NFLABR = READAT (IPOTRA, 47)
```

```
      IF (NFLABR .NE. 0) THEN
*       If the transporter is breakdown
*       Updates the idle statistics
        RTIMES = READAT (IPOTRA, 39)
        NTEMPO = NSTIME()
        RTIMES = RTIMES + NTEMPO - READAT (IPOTRA, 34)
*       ---------------------------------------
        CALL DANENP (IPOTRA, 39, RTIMES)
        CALL DANENP (IPOTRA, 34, REAL(NTEMPO))
*       ---------------------------------------

*       Defines the transporter attribute - Ocupation flag = 0
*       -------------------------------
        CALL DANENP ( IPOTRA, 9, 0.0)
*       -------------------------------

*       Reads the system  attribute -
*                                     pointer to the processing queue
        IPOTPQ = READAT ( IPOSYS, 24)
*       Take the transporter from the processing queue
*       ----------------------------
        CALL TAKENT (IPOTRA, IPOTPQ)
*       ----------------------------
      ELSE
*       If the transporter is still working
*       Defines the transporter attribute - Ocupation flag = -1
*       -------------------------------
        CALL DANENP ( IPOTRA, 9, -1.0)
*       -------------------------------
*         Schedules the event 7  - TRANSPORTER FREE
*               ( ent. TRANSPORTER  t = 0)
*         ============================
          CALL SCHEDU (IPOTRA, 0, 7)
*         ============================
      END IF

      RETURN
      END
```

APPENDIX III.D

ECSL program to simulate a warehouse with three loading bays [Pidd 1988]

SECTIONS

DEFINITIONS

```
THERE ARE 8 TRUCK SET WAIT OS
THERE ARE 15 VAN SET QUEUE WORLD
THERE ARE 6 BAY SET FREE
THERE ARE 1 ZZARRI
THERE ARE 1 ZZCOME
```

INITIALISATION

```
FUNCTION PICTURE NORMAL NEGEXP RANDOM
RECYCLE
RUNINZ= 120 AND PREVCLOCK = RUNINZ
SWITCH ADD ON AFTER RUNINZ
```

ACTIVITIES

RECORDING

```
ACTIVITIES 1008
DURATION= CLOCK - PREVCLOCK
ARRAY ZAWAIT (445)
FOR Z=PREVCLOCK +1 CLOCK
  ADD WAIT TO ZAWAIT ((Z - RUNINZ) / 2 +1)
FOR ZZARRI WITH TIME OF ZZARRI LT 0
  ADD DURATION TO AZZZARRI
FOR ZZCOME WITH TIME OF ZZCOME LT 0
  ADD DURATION TO BZZZCOME
PREVCLOCK = CLOCK
```

ACTIVITY DOCK

```
BEGIN DOCK
FIND FIRST TRUCK A IN WAIT
EXISTS(2) BAY FROM FREE
DURATION= RANDOM(3,SA) + 1
ADD 1 TO DOCK
AADURATION= DURATION + NORMAL(60,5,SE)
TRUCK A FROM WAIT INTO OS AFTER AADURATION
FOR 1 TO 2
  FIND FIRST BAY B IN FREE
  BAY B FROM FREE INTO FREE AFTER AADURATION
REPEAT
```

ACTIVITY MOVE

```
BEGIN MOVE
FIND FIRST VAN A IN QUEUE
FIND FIRST BAY B IN FREE
DURATION= RANDOM( 3, SB ) + 1
ADD 1 TO MOVE
AADURATION= DURATION + NEGEXP (20, SE)
VAN A FROM QUEUE INTO WORLD AFTER AADURATION
BAY B FROM FREE INTO FREE AFTER AADURATION
REPEAT
```

ACTIVITY COME

```
BEGIN COME
TIME OF ZZCOME LE 0
FIND FIRST VAN A IN WORLD
DURATION= NEGEXP (30, SC)
ADD 1 TO COME
VAN A FROM WORLD INTO QUEUE AFTER DURATION
TIME OF ZZCOME = DURATION
```

ACTIVITY ARRIVE

```
BEGIN ARRIVE
TIME OF ZZARRI LE 0
FIND FIRST TRUCK A IN OS
DURATION= NEGEXP ( 120, SD)
TRUCK A FROM OS INTO WAIT AFTER DURATION
TIME OF ZZARRI = DURATION
ADD 1 TO ARRIVE
```

FINALISATION

```
FINALISATION
PRINT 'DOCK  was started' DOCK ' times'
PRINT 'MOVE  was started' MOVE ' times'
PRINT 'COME  was started' COME ' times
PRINT 'ARRIVE was started' ARRIVE ' times'
PRINT 'Utilization of ARRIVE'+4.(1-AZZZARRI/(1.* (CLOCK-RUNINZ)))
PRINT 'Utilization of COME'+4.(1-BZZZCOME/(1.* (CLOCK - RUNINZ)))
```

DATA

```
DATA
OS 1 TO *
WORLD 1 TO *
FREE 1 TO *
SF 5973
SE 3975
SD 7935
SC 7359
SB 3579
SA 5397
END
```

APPENDIX III.E

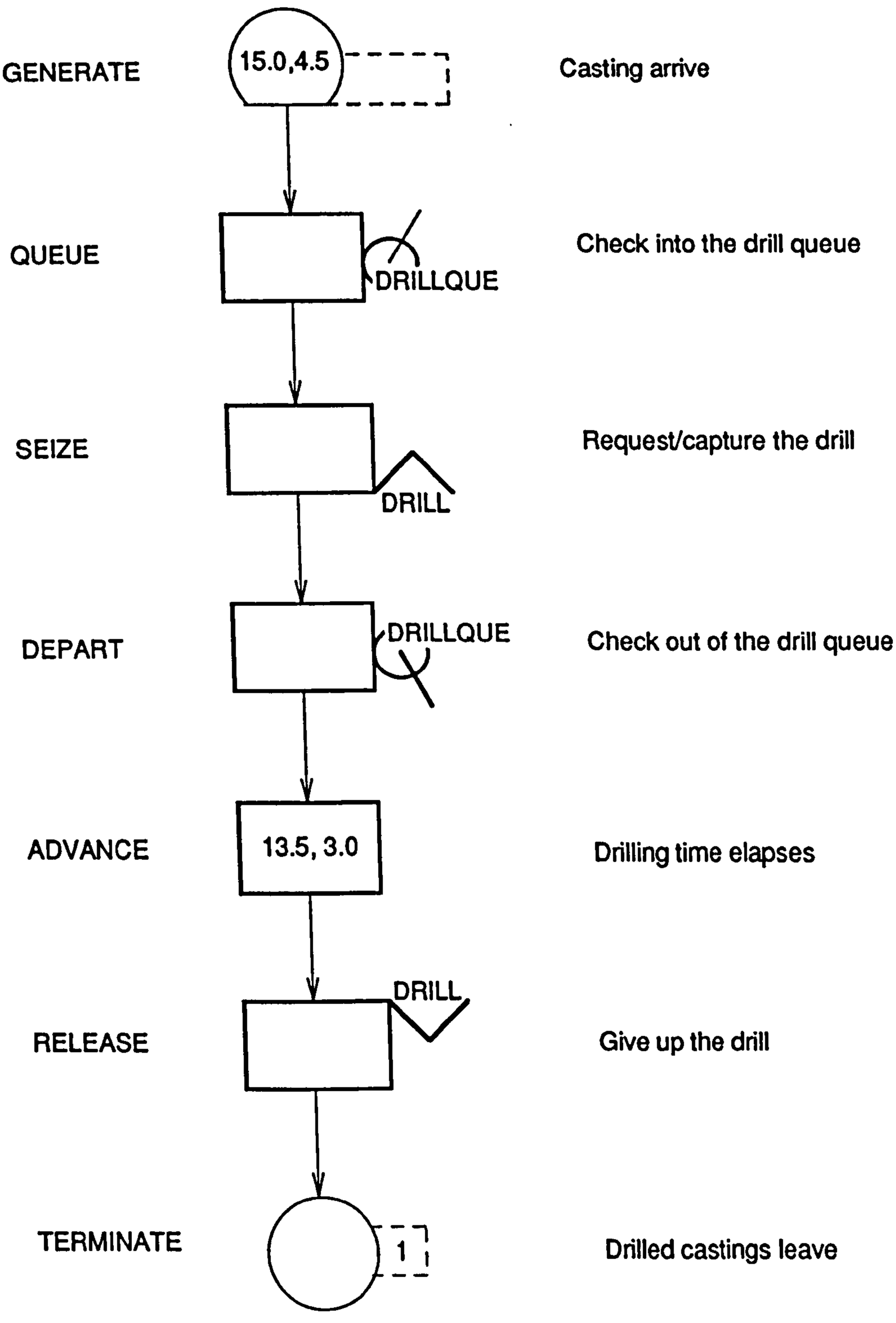
A SIMSCRIPT II.5 Petrol Station Model (Process Approach) [Russell 1989]

```
PREAMBLE
      PROCESS INCLUDE GENERATOR AND CUSTOMER
      RESOURCES INCLUDE ATTENDANT
      ACCUMULATE AVG.QUEUE.LENGTH AS THE AVERAGE
            AND MAX.QUEUE.LENGTH AS THE MAXIMUM
            OF N.Q.ATTENDANT
      ACCUMULATE UTILIZATION AS THE AVERAGE OF N.X.ATTENDANT
END


MAIN
      CREATE EVERY ATTENDANT(1)
      LET U.ATTENDANT(1) = 2
      ACTIVATE A GENERATOR NOW
      START SIMULATION
      PRINT 4 LINES WITH AVG.QUEUE.LENGTH(1), MAX.QUEUE.LENGTH(1),
            AND UTILIZATION(1) * 100. / 2 THUS
            SIMPLE GAS STATION MODEL WITH 2 ATTENDANTS
                  AVERAGE CUSTOMER QUEUE LENGTH IS    *.***
                  MAXIMUM CUSTOMER QUEUE LENGTH IS    *
                  THE ATTENDANTS WERE BUSY **.** PER CENT OF THE TIME.
END


PROCESS   GENERATOR
      FOR I = 1 TO 1000,
      DO
            ACTIVATE A CUSTOMER NOW
            WAIT UNIFORM.F(2.0, 8.0, 1) MINUTES
      LOOP
END


PROCESS   CUSTOMER
      REQUEST 1 ATTENDANT(1)
      WORK UNIFORM.F(5.0, 15.0, 2) MINUTES
      RELINQUISH 1 ATTENDANT(1)
END
```

```
PREAMBLE
      EVENT NOTICES INCLUDE ARRIVAL
            EVERY END.SERVICE HAS AN ES.CUSTOMER
      TEMPORARY ENTITIES
            EVERY CUSTOMER HAS AN TIME.OF.ARRIVAL
                  AND MAY BELONG TO THE QUEUE
      THE SYSTEM OWNS THE QUEUE
      DEFINE NO.BUSY AND NO.CUSTOMERS AS INTEGER VARIABLES
      ACCUMULATE AVG.QUEUE.LENGTH AS THE AVERAGE
            AND MAX.QUEUE.LENGTH AS THE MAXIMUM
            OF N.QUEUE
      ACCUMULATE UTILIZATION AS THE AVERAGE OF NO.BUSY
END


MAIN
      SCHEDULE A ARRIVAL NOW
      START SIMULATION
      PRINT 4 LINES WITH AVG.QUEUE.LENGTH, MAX.QUEUE.LENGTH,
            AND UTILIZATION(1) * 100./2 THUS
            SIMPLE GAS STATION MODEL WITH 2 ATTENDANTS
                  AVERAGE CUSTOMER QUEUE LENGTH IS    *.***
                  MAXIMUM CUSTOMER QUEUE LENGTH IS    *
                  THE ATTENDANTS WERE BUSY **.** PER CENT OF THE TIME.
END


EVENT  ARRIVAL
      IF NO.CUSTOMERS < 1000,
            ADD 1 TO NO.CUSTOMERS
            CREATE A CUSTOMER
            LET TIME.OF.ARRIVAL(CUSTOMER) = TIME.V
            IF NO.BUSY = 2,
                  FILE CUSTOMER IN QUEUE
            ELSE
                  ADD 1 TO NO.BUSY
                  SCHEDULE AN END.SERVICE GIVING CUSTOMER
                        IN UNIFORM.F(5.0, 15.0, 2) MINUTES
            ALWAYS
            SCHEDULE A ARRIVAL IN UNIFORM.F(2.0, 8.0, 1) MINUTES
      ALWAYS
END


EVENT  END.SERVICE  GIVEN  CUSTOMER
      DEFINE CUSTOMER AS AN INTEGER VARIABLE
      LET SERVICE.TIME = TIME.V - TIME.OF.ARRIVAL(CUSTOMER)
      DESTROY CUSTOMER
      IF QUEUE IS NOT EMPTY,
            REMOVE THE FIRST CUSTOMER FROM QUEUE
            SCHEDULE AN END.SERVICE GIVING CUSTOMER
                  IN UNIFORM.F(5.0, 15.0, 2) MINUTES
      ELSE
            SUBTRACT 1 FROM NO.BUSY
      ALWAYS
END
```

APPENDIX III.F

GPSS Block Diagram for a One-Line, One-Server Queueing System [Schriber 1987]

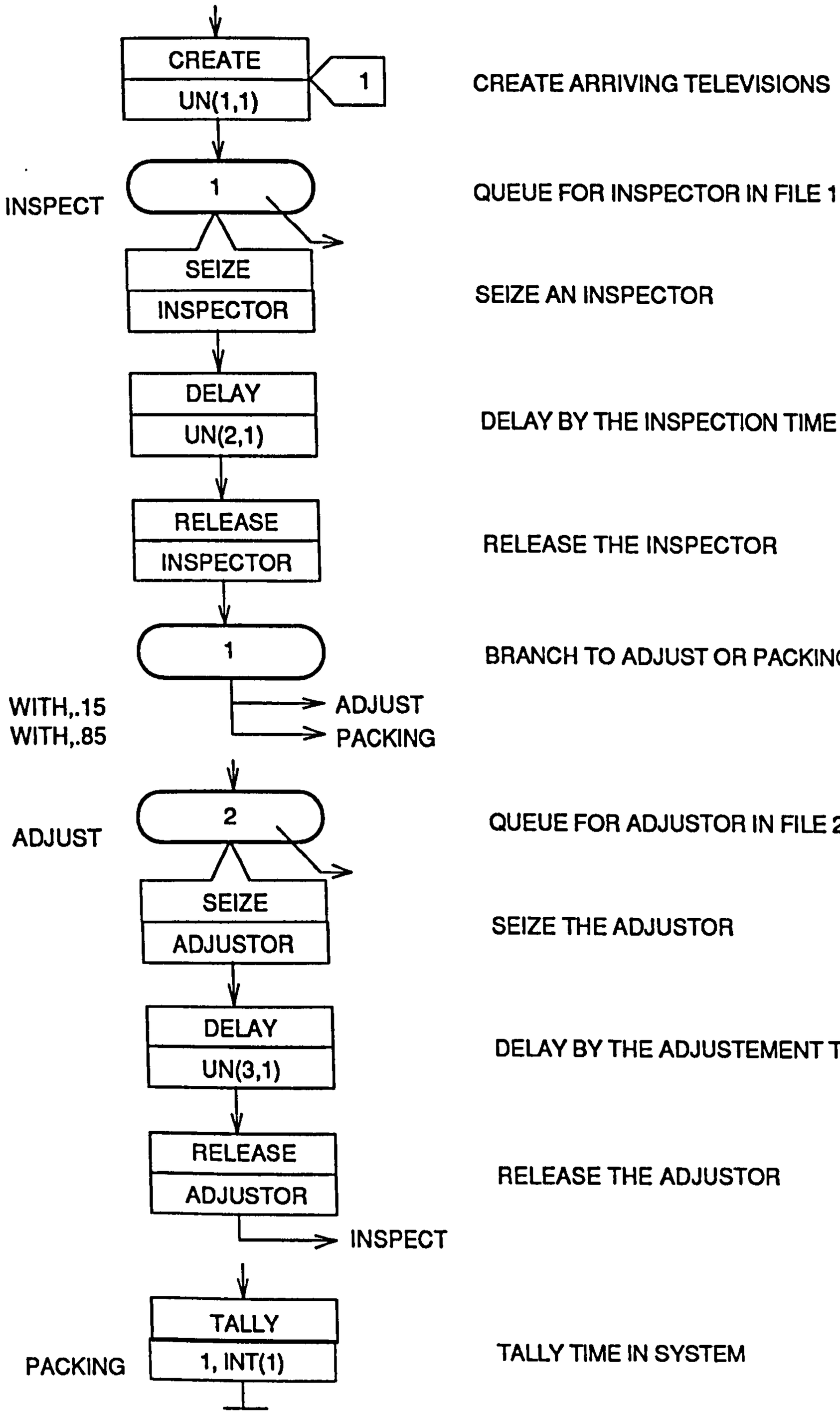


A GPSS Model for a One-Line, One-Server Queueing System [Schriber 1987]

```
SIMULATE
*
**********************************************************************
*          Model Segment 1 (Movement of Castings Through the System
**********************************************************************
*
GENERATE      15.0, 4.5     Casting arrives
QUEUE         DRILLQUE      Check into the drill queue
SEIZE         DRILL         Request/capture the drill
DEPART        DRILLQUE      Check out of the drill queue
ADVANCE       13.5, 3.0     Drilling time elapses
RELEASE       DRILL         Give up the drill
TERMINATE     1             Drilled castings leave
*
**********************************************************************
*          Run-Control Statments
**********************************************************************
*
              START         100           Start the simulation
*
              END                         End the simulation
```

APPENDIX III.G

SIMAN Block Diagram for a TV Inspection System [Pedgen 1984]



SIMAN Model for a TV Inspection System [Pedgen 1984]

MODEL FRAME

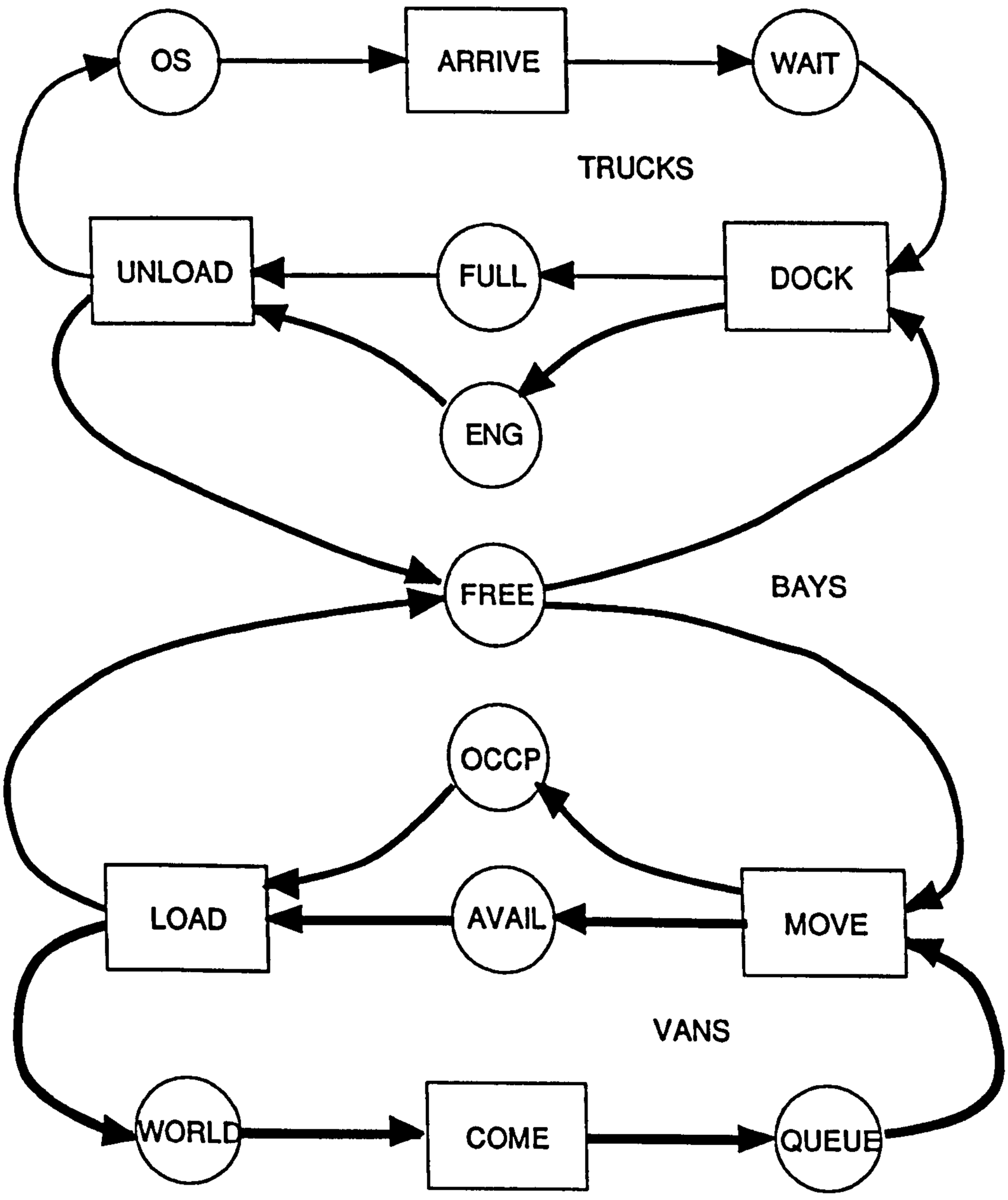
```
BEGIN;
10                CREATE: UN(1,1): MARK(1);          CREATE ARRIVING TELEVISIONS
20   INSPECT      QUEUE,1;                           QUEUE FOR INSPECTOR IN FILE 1
30                SEIZE: INSPECTOR;                  SEIZE AN INSPECTOR
40                DELAY: UN(2,1);                    DELAY BY THE INSPECTION TIME
50                RELEASE: INSPECTOR;                RELEASE THE INSPECTOR
60                BRANCH,1:
                      WITH,.15,ADJUST:
                      WITH,.85,PACKING;              BRANCH TO ADJUST OR PACKING
70   ADJUST       QUEUE,2;                           QUEUE FOR ADJUSTOR IN FILE 2
80                SEIZE: ADJUSTOR;                   SEIZE THE ADJUSTOR
90                DELAY: UN(3,1);                    DELAY BY THE ADJUSTEMENT TIME
100               RELEASE: ADJUSTOR: NEXT(INSPECT);  RELEASE THE ADJUSTOR
110  PACKING      TALLY:1,INT(1):DISPOSE;            TALLY TIME IN SYSTEM
     END;
```

EXPERIMENTAL FRAME

```
     BEGIN;
10   PROJECT, TV INSP AND ADJT, PEDGEN, 4/25/81;
20   DISCRETE, 30, 1, 2;
30   PARAMETERS: 1, 3.5, 7.5,: 2, 6., 12: 3, 20, 40;
40   RESOURCES: 1, INSPECTOR, 2: 2, ADJUSTOR;
50   TALLIES: 1, TIME IN SYSTEM;
60   DSTAT: 1, NQ(1), INSP QUEUE:2, NQ(2), ADJT QUEUE:
            3, NR(1), INSP UTIL:4, NR(2), ADJT UTIL;
70   TRACE;
80   REPLICATE, 1, 0, 100;
     END;
```

APPENDIX III.H

The Activity Cycle Diagram for a Warehouse system [Pidd 1988]



The CAPS Dialogue for a Warehouse System Model [Pidd 1988]

EXTENDED CONTROL AND SIMULATION LANGUAGE
COMPUTER AIDED PROGRAMMING SYSTEM

Do you wish to have instructional comments? N
Problem name? WAREHOUSE
Do you wish to start a new problem? Y

LOGIC
Type name of one kind of entity? TRUCK
How many? 8
Type a list of the states through which these entities pass
precede queues by Q and activities by A
?QOS, AARRIVE, QWAIT, ADOCK, QFULL, AUNLOAD, QOS
Is this cycle correct? Y
Type name of one kind of entity? VAN
How many? 15
Type a list of the states through which these entities pass
precede queues by Q and activities by A
?QWORLD, ACOME, QQUEUE, AMOVE, QAVAIL, ALOAD, QWORLD
Is this cycle correct? Y
Type name of one kind of entity? BAY
How many? 6
Type a list of the states through which these entities pass
precede queues by Q and activities by A
?QFREE, ADOCK, QENG, AUNLOAD, QFREE, AMOVE, QOCCP, ALOAD, QFREE
Is this cycle correct? Y
Type name of one kind of entity?

Are there any (other) activities which use more than one entity of a particular type? Y
Which activity? DOCK
Which entity type? BAY
How many entities per activity? 2

Which activity? UNLOAD
Which entity type? BAY
How many entities per activity? 2

Which activity?

From what you have said so far, the following are the maximum
number of simultaneous realisations of the activities.

Activity	number	
ARRIVE	8	Limited by the number of TRUCK
DOCK	3	Limited by the number of BAY
UNLOAD	3	Limited by the number of BAY
COME	15	Limited by the number of VAN
MOVE	6	Limited by the number of BAY
LOAD	16	Limited by the number of BAY

Do you wish to apply limits which are below these? Y
Which activity? ARRIVE
What is the limit? 1

Which activity? COME
What is the limit? 1

Which activity?

Not more than 7 of the 8 TRUCK can be active at one time
Not more than 13 of the 15 VAN can be active at one time
Activity UNLOAD appears to be bound to DOCK
i.e. the following queues are dummies
FULL
ENG
Do you agree? Y

Activity LOAD appears to be bound to MOVE
i.e. the following queues are dummies
AVAIL
OCCP
Do you agree? Y

Do you wish to see a summary of the cycles? Y
TRUCK 8, QOS, AARRIVE, QWAIT, ADOCK, Q, AUNLOAD, QOS
VAN 15, QWORLD, ACOME, QQUEUE, AMOVE, Q, ALOAD, QWORLD
BAY 6, QFREE, ADOCK, Q, AUNLOAD, QFREE, AMOVE, Q, ALOAD, QFREE
ZZARRI 1, AARRIVE
ZZCOME 1, ACOME

ARRIVE	USES	1	TRUCK	1	ZZARRI
DOCK	USES	1	TRUCK	2	BAY
UNLOAD	USES	1	TRUCK	2	BAY
COME	USES	1	VAN	1	ZZCOME
MOVE	USES	1	VAN	1	BAY
LOAD	USES	1	VAN	1	BAY

Do you wish to make any changes in the logic section? N

PRIORITIES
Are there any queues whose discipline is not FIFO? N
The following are bound activities
UNLOAD
LOAD

The order of the following activities is unimportant
COME
ARRIVE

I propose putting the remaining activities in the following order
MOVE
DOCK
Do you wish to raise the priority of any activity? Y
Which activity? DOCK
Which activity?

Do you wish to make any changes in the priority section? N

ARITHMETIC
After each activity name, type formula for its duration if the duration might be zero, type 0+....
DOCK = ? RANDOM(3, SA) + 1
MOVE = ? RANDOM(3, SB) + 1
COME = ? NEGEXP(30, SC)
ARRIVE = ? NEGEXP(120, SD)
UNLOAD = ? NORMAL(60, 5, SE)
LOAD = ? NEGEXP(20, SF)
In which activity is SA evaluated?
What is its initial value? 5397
In which activity is SB evaluated?
What is its initial value? 3579
In which activity is SC evaluated?
What is its initial value? 7359
In which activity is SD evaluated?
What is its initial value? 7935
In which activity is SE evaluated?
What is its initial value? 3975
In which activity is SF evaluated?
What is its initial value? 5973
Do you wish to define any more attributes for entities? N
Do you wish to make any changes to the arithmetic section? N

RECORDING
WAIT = ? 3
OS = ? 0
QUEUE = ? 3
WORLD = ? 0
FREE = ? 3
What length of run-in period is required? 120
Do you wish to make any changes to the recording section? N

INITIAL CONDITIONS
Are there any activities in progress? N

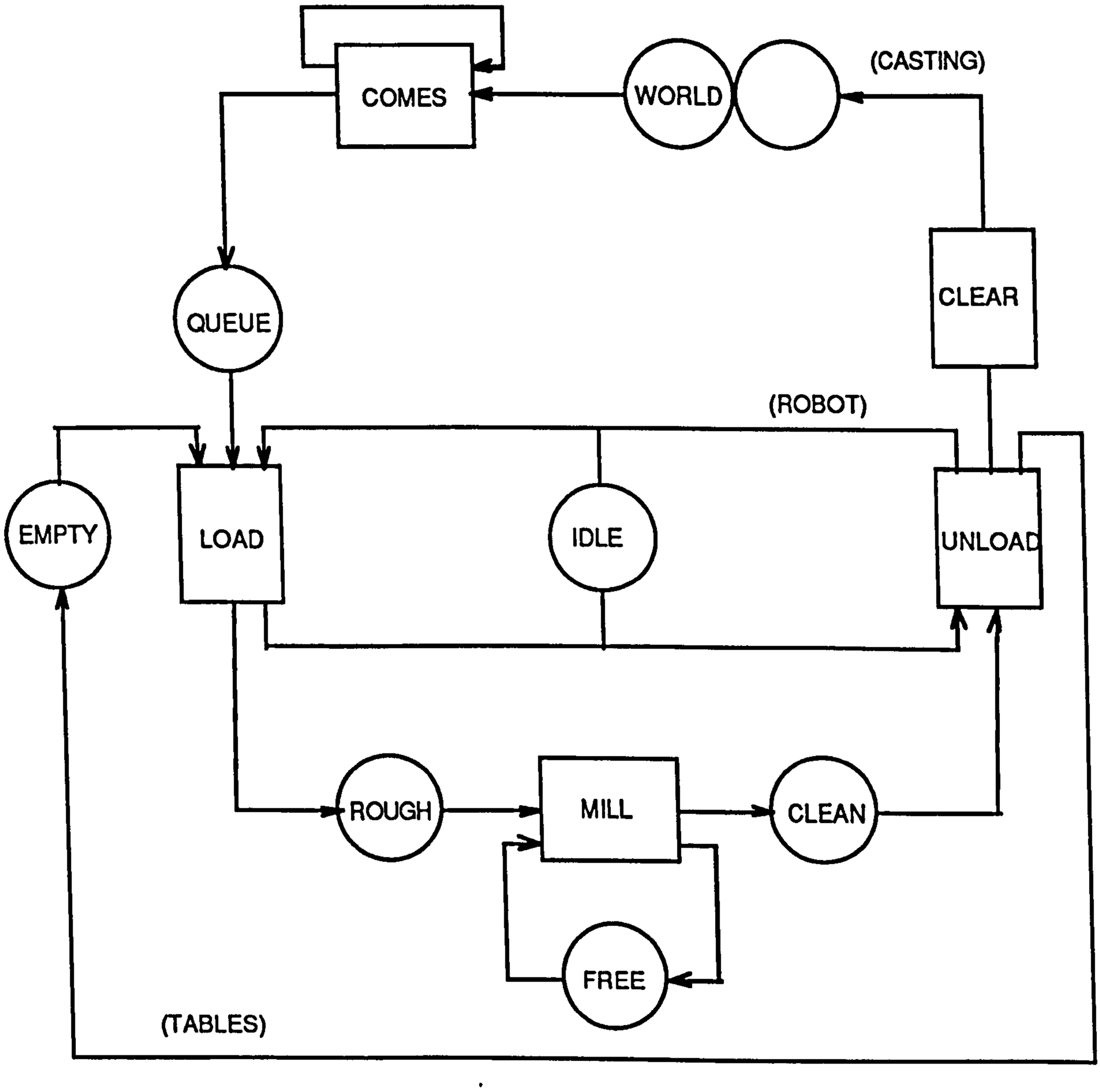
Type how many entities should be in each queue listed after the queue name
TRUCK - 8 Entities
WAIT ? 0
OS ? 8
VAN - 15 Entities
QUEUE ? 0
WORLD ? 15
BAY - 6 Entities
6 Entities placed in queue FREE
Please give the duration of the simulation? 1008
Do you wish to make any changes in the initial condition section? N

LOAD, which you have used as a name is an ECSL keyword
please give a replacement? FILL

Some entities are apparently suitable for aggregation
i.e. they have no attributes, no delay recording and use only FIFO Q-discipline
Do you wish me to aggregate TRUCK? N
Do you wish me to aggregate VAN? N
Do you wish me to aggregate BAY? N
Your program has been written have you finished? Y

APPENDIX III.I

The Activity Cycle Diagram for a Robot Cell [Mathewson 1989]



The DRAFT Dialogue for the Robot Cell Model [Mathewson 1989]

```
THIS IS DRAFT. LICENDED TO:-PAXMAN        V. 9/4(86
INPUT IS FREE FORMAT. DATA  IS SEPARATED BY A BLANK OR COMMA.
MODE IS "KEYIN" OR "EDIT". A "SOURCE" FILE MUST BE SPECIFIED
FOR "EDIT" MODE (FIRST OR LAST). NOW ENTER:- MODE, SCRATCH FILE (OPTIONAL)

E.G. "KEYIN" OR "EDIT,LAST" ? k
PLEASE DEPRESS THE CAPS LOCK
AND REPEAT THE INPUT IN UPPER CASE
K(EYIN) OR E(DIT, (F(IRST) OR L(AST)) ? K

INPUT FILE NAME "XXXXXXXX" FOR PROGRAM.
IT WILL BE ON THE FILE CALLED "XXXXXXXX.FOR". ? WILEY

PLEASE ENTER THE NAMES OF ACTIVITIES IN PRIORITY ORDER
",+" INDICATES MORE NAMES ON THE NEXT LINE. * FOR NO DATA
? UNLOAD MILL LOADS COMES CLEAR

NOW THE QUEUE NAMES. * IF NONE
? WORLD QUEUE ROUGH CLEAN EMPTY FREE IDLE

PAGE 3. ENTER THE DURATION FOR EACH ACTIVITY, IN THE ORDER
THE ACTIVITIES WERE DEFINED. FINALLY TOTAL RUN TIME.
? 1 5 6 7 8 100
STOCHASTIC VALUES REQUIRED PLEASE ENTER DETAILS
NORMAL CODE= 1, STREAM, MEAN, SD
NEGEX CODE= 2, STREAM, MEAN
UNIFORM CODE= 3, STREAM, LOWER, UPPER
SAMPLE CODE= 4, STREAM
DEFINE DATA FOR ACTIVITY UNLOA
? 2  1  10.

PAGE 4. NOW ENTER CYCLES.
THE ENTITY NAME "ERROR" WILL ENTER EDIT MODE.

CYCLE 1.
NAME THE ENTITY CLASS, SIZE, AND "H" FOR HISTOGRAM
? INPUT  1
ENTITY CYCLE HISTORY
? COMES

CYCLE 2.
NAME THE ENTITY CLASS, SIZE, AND "H" FOR HISTOGRAM
? CASTING  100  H
ENTITY CYCLE HISTORY
? WORLD COMES QUEUE LOADS ROUGH MILL CLEAN UNLOAD CLEAR
ENTER HISTOGRAM DATA
ENTITY DELAYS - GIVE MEASUREMENT POINTS E.G. "END COMES, START SERVE"
QUEUE SIZE - GIVE QUEUE NAME
? E  COMES  E  UNLOAD
NOW ENTER LOWER BOUND AND STEP
? 5  5
```

```
CYCLE 3.
NAME THE ENTITY CLASS, SIZE, AND "H" FOR HISTOGRAM
? ROBOT 1
ENTITY CYCLE HISTORY
? IDLE LOADS IDLE UNLOAD

CYCLE 4.
NAME THE ENTITY CLASS, SIZE, AND "H" FOR HISTOGRAM
? HEAD 1
ENTITY CYCLE HISTORY
? FREE MILL

CYCLE 5.
NAME THE ENTITY CLASS, SIZE, AND "H" FOR HISTOGRAM
? TABLES 2
ENTITY CYCLE HISTORY
? EMPTY LOADS ROUGH MILL CLEAN UNLOAD

CYCLE 6.
NAME THE ENTITY CLASS, SIZE, AND "H" FOR HISTOGRAM
? FINIS

ENTER ZERO TO CONTINUE - PAGE NUMBER TO EDIT? 0
PLEASE SPECIFY: -
THE TIME STRUCTURE          EVENT: THREE-PHASE ,
THE CLOCK                   REAL: INTEGER, AND
OPTIMISED CODE              YES: NO
? E R Y
LOADS  MAY  START  FROM  UNLOA
UNLOA  MAY  START  FROM  UNLOA
LOADS  MAY  START  FROM  UNLOA
UNLOA  MAY  START  FROM  MILL
MILL   MAY  START  FROM  MILL
MILL   MAY  START  FROM  LOADS
LOADS  MAY  START  FROM  LOADS
UNLOA  MAY  START  FROM  LOADS
LOADS  MAY  START  FROM  COMES
ENTER ZERO TO CONTINUE - PAGE NUMBER TO EDIT ? 0

DATA INPUT OK, PROG ON WILEY.FOR - DATA ON WILEY.DAT
********** THE MEMORY IS NOW LAST **********
```

APPENDIX IV.A

A Warehouse Simulation Model Using PC-Model

```
;Brito, A.E.C., 881028

W=(50)
O=(=)
D=
```

WAREHOUSE SIMULATION SYSTEM

This is a model of a warehouse with one floor high racking and full pallete in full pallete out.

```
;Brito, A.E.C., 881028
$


;Initializes some variables
@@DIF=(10,0)                    ;Aux. variable
@DIFF=(0)
@I=(0)
@J=(0)
@IC=(0)
@TMP=(0)
@TMP1=(0)
@XE=(0)
@YE=(0)
@NE=(0)
@NCE=(0)
@NC=(0)
@IX=(0)
@TOTPAL=(0)
@SUM=(0)
@II=(0)
@IFLAG=(0)
@@FAC=(10,0)
@PN=(0)
@DELAY=(0)
@PARFLA=(0)
@IQ=(0)
@JQ=(0)
@KQ=(0)
@IU=(0)
@IO=(0)
@PP=(0)
@PALORD=(0)
@JC=(0)
@XX=(0)
@YY=(0)
@KK=(0)
@LL=(0)
@JJ=(0)
@KFLAG=(0)
@NFRECE=(0)
```

```
@FILPER=(60)          ;Fill percentage
@NUMPRO=(4)            ;Number of products
@TIMPIC=(15)          ;Time delay to pick from a cell
@INTART=(1800)         ;Outorders inter-arrival time
@PALOME=(15)           ;Number of pallets/outorder (mean)
@PALOSD=(3)           ;Number of pallets/outorder (SD)
@NOUTWA=(0)            ;Number of outorders waiting
@TIMREC=(30)          ;Time delay to pick in the reception
@NINOWA=(0)           ;Number of inorders waiting

*CELL=(XY(1,1))
*TRAC=(XY(1,1))
*DIPO=(XY(1,1))

;Defines the vectors and arrays

@NUMREC=(3)           ;Number of reception points
@@LOCREC=(3,2)          ;Reception points location
@@NPROPA=(4,132)        ;Number of the product pallet cells
@@FREECE=(132,0)        ;Free cells index
@@PRICEL=(132,0)       ;Priority cell vector
@@PROROT=(4,0)         ;Percentage product rotation
@@LOCDES=(3,2)          ;Despatch points location
@@DESFLA=(3,0)         ;Despatch points ocupation flag
@@RECFLA=(3,0)          ;Reception points ocupation flag
@@PICTCO=(2,2)         ;Pick trucks park coordinates
@@REATCO=(2,2)         ;Reach trucks park coordinates
@@DESQUE=(3,0)         ;Queue of outorders
@@ATPICT=(2,5)         ;Pick truck attributes
@@NUPADE=(3,4)          ;Number of pallets in the despatch
@@MINPRO=(4,0)         ;Minimum number of pallets for reorder
@@NUPARO=(4,0)          ;Number of pallets for inordering
@@TIDERO=(4,0)         ;Time delay for inorder to arrive
@@ATREAT=(2,5)          ;Reach truck attributes
@@RECQUE=(3,0)          ;Queue of inorders
@@NPROIN=(4,0)         ;Inorder product number
@@NUPARE=(3,2)          ;Number of pallets in the reception
@@INOFLA=(4,0)         ;Flag for inordering
@@NOUTSI=(4,0)          ;Number of out of stock situations
@@NPALOS=(4,0)          ;Number of pallets out of stock

;Jobs definition

J=(1, ,1,0,0,0,1)
J=(2, ,2,0,0,8,2)
J=(3, ,3,0,0,9,1)
J=(4, ,4,0,0,8,2)
;J=(5, ,5,0,0,9,1)

;Transporters utilisation statistics definition

U=(1,PICK 1,XY(63,2))
U=(2,PICK 2,XY(65,21))
U=(3,REACH 1,XY(13,2))
U=(4,REACH 2,XY(15,21))

;Uses this route for variables initialization
```

```
BR(1,XY(1,1),0)

   ;Defines the reception points coordinates

   PO(XY(1,15))
   SA(0,7)
   SV(@@LOCREC(1,1),8)
   SV(@@LOCREC(1,2),4)
   SV(@@LOCREC(2,1),8)
   SV(@@LOCREC(2,2),11)
   SV(@@LOCREC(3,1),8)
   SV(@@LOCREC(3,2),18)
   SV(@FILPER,60)

      ;Reception points ocupation flag initialization

   SV(@@RECFLA(1,0),0)
   SV(@@RECFLA(2,0),0)
   SV(@@RECFLA(3,0),0)

      ;Defines the despatch points location

   SV(@@LOCDES(1,1),70)
   SV(@@LOCDES(1,2),5)
   SV(@@LOCDES(2,1),70)
   SV(@@LOCDES(2,2),12)
   SV(@@LOCDES(3,1),70)
   SV(@@LOCDES(3,2),19)

      ;Despatch points ocupation flag initialization

   SV(@@DESFLA(1,0),0)
   SV(@@DESFLA(2,0),0)
   SV(@@DESFLA(3,0),0)

      ;Initialize the inorders queue

   SV(@@RECQUE(0,0),0)
   SV(@@RECQUE(1,0),0)
   SV(@@RECQUE(2,0),0)
   SV(@@RECQUE(3,0),0)

      ;Initialize the inorders flag

   SV(@@INOFLA(1,0),0)
   SV(@@INOFLA(2,0),0)
   SV(@@INOFLA(3,0),0)
   SV(@@INOFLA(4,0),0)

      ;Initialize the outorders queue

   SV(@@DESQUE(0,0),0)
   SV(@@DESQUE(1,0),0)
   SV(@@DESQUE(2,0),0)
   SV(@@DESQUE(3,0),0)

      ;Initialize the number of out of stock situations variable

   SV(@@NOUTSI(1,0),0)
```

```
SV(@@NOUTSI(2,0),0)
SV(@@NOUTSI(3,0),0)
SV(@@NOUTSI(4,0),0)

   ;Initialize the number of pallets out of stock

SV(@@NPALOS(1,0),0)
SV(@@NPALOS(2,0),0)
SV(@@NPALOS(3,0),0)
SV(@@NPALOS(4,0),0)

   ;Defines the pick trucks park coordinates

SV(@@PICTCO(1,1),63)
SV(@@PICTCO(1,2),2)
SV(@@PICTCO(2,1),65)
SV(@@PICTCO(2,2),21)

   ;Defines the reach trucks park coordinates

SV(@@REATCO(1,1),13)
SV(@@REATCO(1,2),2)
SV(@@REATCO(2,1),15)
SV(@@REATCO(2,2),21)

   ;Prepares the free cell vector

SV(@IC,133)
SV(@I,22)
:LOOP0
 SV(@J,1)
 SV(@SUM,@I)
 :LOOP3
  DV(@IC)
  SV(@@FREECE(@IC,0),@SUM)
  SV(@@PRICEL(@SUM,0),@IC)
  AO(@SUM,+,22)
  IV(@J)
 IF(@J,LE,6,THEN,:LOOP3)
 DV(@I)
IF(@I,GE,1,THEN,:LOOP0)
SV(@@FREECE(0,0),132)

   ;Reads the initial fill percentage

PM(XY(18,19),Warehouse fill percentage)
GV(@FILPER)
PM(XY(18,19),                    )

   ;Defines the product rotation

SV(@@PROROT(1,0),60)          ;Product A 60%
SV(@@PROROT(2,0),20)          ;Product B 20%
SV(@@PROROT(3,0),10)          ;Product C 10%
SV(@@PROROT(4,0),10)          ;Product D 10%

   ;Defines the minimum number of pallets for generate a inorder

SV(@@MINPRO(1,0),40)
SV(@@MINPRO(2,0),13)
```

```
SV(@@MINPRO(3,0),7)
SV(@@MINPRO(4,0),7)

   ;Defines the number of pallets per inorder

SV(@@NUPARO(1,0),40)
SV(@@NUPARO(2,0),13)
SV(@@NUPARO(3,0),7)
SV(@@NUPARO(4,0),7)

   ;Defines the lead time for the inorder to arrive

SV(@@TIDERO(1,0),600)
SV(@@TIDERO(2,0),600)
SV(@@TIDERO(3,0),600)
SV(@@TIDERO(4,0),600)

    ;Defines the initial fill

SV(@TOTPAL,@FILPER)
AO(@TOTPAL,*,132)
AO(@TOTPAL,/,100)
SV(@I,1)
:LOOP1
  SV(@@NPROPA(@I,0),@@PROROT(@I,0))
  AO(@@NPROPA(@I,0),*,@TOTPAL)
  AO(@@NPROPA(@I,0),/,100)
  SV(@IX,@@NPROPA(@I,0))
  SV(@J,1)
  :LOOP2
    SV(@IC,@@FREECE(0,0))
    SV(@NC,@@FREECE(@IC,0))
    DV(@@FREECE(0,0))
    SV(@@NPROPA(@I,@IX),@NC)
    LK(!COORD)
    SV(@PN,@I)
    LK(!PUPROC)
    DV(@IX)
    IV(@J)
  IF(@J,LE,@@NPROPA(@I,0),THEN,:LOOP2)
  IV(@I)
IF(@I,LE,@NUMPRO,THEN,:LOOP1)

;Reads the inter-arrival time, mean number of pallets and SD

PM(XY(18,19),Outorders inter-arrival time)
GV(@INTART)
   PM(XY(18,19),                    )
   PM(XY(18,19),Average pallets per outorder)
   GV(@PALOME)
   PM(XY(18,19),                    )
   PM(XY(18,19),S.D of pallets/outorder)
   GV(@PALOSD)
   PM(XY(18,19),                    )
ER

      ;Defines the route for the transporters job 1: Loading

BR(2,XY(1,1),0)
```

```
   ;Initialize the job attributes

SV(OBJ@3,0)                ;Destin xx coordinate
SV(OBJ@4,0)                ;Destin yy coordinate
SV(OBJ@6,4)                 ;Position flag

   ;Position the transporter in the park

LK(!PIPA)
:BEGIN
SV(@IFLAG,1)
IF(OBJ@5,EQ,0,THEN,NEXT,ELSE,:ENDIF12)

  ;If the transporter is not working yet

  IF(@@DESQUE(0,0),NE,0,THEN,NEXT,ELSE,:ENDIF12)

    ;If there is any outorder waiting

    SV(@KK,1)
    :LOOP9
     SV(@LL,@@DESQUE(@KK,0))
     SV(@JJ,1)
     :LOOP10
      IF(@@NUPADE(@LL,@JJ),NE,0,THEN,NEXT,ELSE,:ENDIF29)
        SV(OBJ@5,@LL)
        SV(@KK,@@DESQUE(0,0))
        SV(@JJ,@NUMPRO)
      :ENDIF29
       IV(@JJ)
      IF(@JJ,LE,@NUMPRO,THEN,:LOOP10)
      IV(@KK)
    IF(@KK,LE,@@DESQUE(0,0),THEN,:LOOP9)
:ENDIF12
IF(OBJ@5,NE,0,THEN,NEXT,ELSE,:ENDIF14)

  ;If the transporter has work to do

  SV(@IFLAG,1)
  SV(@II,1)
  :LOOP4
   IF(@@NUPADE(OBJ@5,@II),NE,0,THEN,NEXT,ELSE,:ENDIF13)
     IF(@@NPROPA(@II,0),GT,0,THEN,NEXT,ELSE,:ELSE80)
      SV(@@ATPICT(OBJ@SN,1),@II)
      SV(@IX,@@NPROPA(@II,0))
      DV(@@NPROPA(@II,0))
      SV(@@ATPICT(OBJ@SN,2),@@NPROPA(@II,@IX))
      SV(@@NPROPA(@II,@IX),0)
     DV(@@NUPADE(OBJ@5,@II))
      SV(@IFLAG,0)
      SV(@II,@NUMPRO)
      JP(:ENDIF80)
    :ELSE80
     IV(@@NOUTSI(@II,0))
     AO(@@NPALOS(@II,0),+,@@NUPADE(OBJ@5,@II))
     SV(@@NUPADE(OBJ@5,@II),0)
    :ENDIF80
  :ENDIF13
  IV(@II)
IF(@II,LE,@NUMPRO,THEN,:LOOP4)
```

```
:ENDIF14
IF(@IFLAG,EQ,0,THEN,NEXT,ELSE,:ELSE16)

 ;Pick the pallet in the cell and bring it to the despatch point

 LK(!PITOCE)
 SV(OBJ@3,@@LOCDES(OBJ@5,1))
 SV(OBJ@4,@@LOCDES(OBJ@5,2))
 LK(!MOVE)
  SV(OBJ@ID," )
 DV(@@NUPADE(OBJ@5,0))
SV(@TMP,@@LOCDES(OBJ@5,2))
 AO(@TMP,-,1)
 AO(@TMP,*,80)
 AO(@TMP,*,2)
SV(@TMP1,@@LOCDES(OBJ@5,1))
 AO(@TMP1,+,2)
 AO(@TMP1,*,2)
 AO(@TMP,+,@TMP1)
 SV(*DIPO,@TMP)
PV(*DIPO,@@NUPADE(OBJ@5,0))
IF(@@NUPADE(OBJ@5,0),EQ,0,THEN,NEXT,ELSE,:ENDIF20)

  ;If the outorder is finished

   PM(*DIPO,+++++)
  SV(@IQ,1)
  :LOOP5
    IF(OBJ@5,EQ,@@DESQUE(@IQ,0),THEN,NEXT,ELSE,:ENDIF25)
     SV(@JQ,@IQ)
   :ENDIF25
   IV(@IQ)
IF(@IQ,LE,@@DESQUE(0,0),THEN,:LOOP5)
DV(@@DESQUE(0,0))
 :LOOP6
IF(@JQ,LE,@@DESQUE(0,0),THEN,NEXT,ELSE,:EXIT6)
  SV(@KQ,@JQ)
  IV(@KQ)
  SV(@@DESQUE(@JQ,0),@@DESQUE(@KQ,0))
  IV(@JQ)
   JP(:LOOP6)
 :EXIT6
 SV(@@DESFLA(OBJ@5,0),0)
 SV(OBJ@5,0)
 IF(@NOUTWA,GT,0,THEN,NEXT,ELSE,:ENDIF20)
  DV(@NOUTWA)
   IF(@NOUTWA,EQ,0,THEN,NEXT,ELSE,:ELSE55)
     PM(XY(68,22),+++++++++)
    JP(:ENDIF55)
   :ELSE55
    PV(XY(73,22),@NOUTWA)
   :ENDIF55
  LK(!OUTARR)
 :ENDIF20
 MU(1,1)
 AO(OBJ@2,-,1)
 SV(OBJ@6,2)
 JP(:ENDIF16)
:ELSE16
 IF(OBJ@5,EQ,0,THEN,NEXT,ELSE,ELSE,:ELSE28)
```

```
;Returns to park

 SV(@PARFLA,1)
IF(@@PICTCO(OBJ@SN,1),NE,OBJ@1,THEN,NEXT,ELSE,:ENDIF21)
  SV(@PARFLA,0)
 :ENDIF21
IF(@@PICTCO(OBJ@SN,2),NE,OBJ@2,THEN,NEXT,ELSE,:ENDIF22)
  SV(@PARFLA,0)
 :ENDIF22
 IF(@PARFLA,EQ,0,THEN,NEXT,ELSE,:ENDIF23)

  ;Returns to park

  SV(OBJ@3,@@PICTCO(OBJ@SN,1))
  SV(OBJ@4,@@PICTCO(OBJ@SN,2))
  IF(OBJ@SN,EQ,1,THEN,NEXT,ELSE,:ELSE24)
    AO(OBJ@3,+,2)
    JP(:ENDIF24)
   :ELSE24
    AO(OBJ@3,-,2)
   :ENDIF24
   LK(!MOVE)
   LK(!PIPA)
  :ENDIF23
  WE
  JP(:ENDIF28)
:ELSE28
  SV(OBJ@5,0)
   :ENDIF28
 :ENDIF16
 JP(:BEGIN)
ER

;Defines the route for outorders arrival

BR(3,XY(1,1),0)
  :BEGOUT
  RV(E,@DELAY,@INTART)
  WT(@DELAY)
  IF(@@DESQUE(0,0),LT,3,THEN,NEXT,ELSE,:ELSE27)

    ;If there is a free despatch point

    LK(!OUTARR)
    JP(:ENDIF27)
   :ELSE27
    IV(@NOUTWA)
    IF(@NOUTWA,EQ,1,THEN,NEXT,ELSE,:ENDIF56)
      PM(XY(68,22),O.W.:     )
     :ENDIF56
     PV(XY(73,22),@NOUTWA)
  :ENDIF27
  SE
 JP(:BEGOUT)
ER

    ;Defines the route for the transporters job 2: Unloading

BR(4,XY(1,1),0)
```

```
   ;Initialize the job attributes

SV(OBJ@3,0)                ;Destin xx coordinate
SV(OBJ@4,0)                ;Destin yy coordinate
SV(OBJ@6,4)                 ;Position flag

   ;Position the transporter in the park

LK(!REPA)
:RBEGIN
SV(@KFLAG,1)
IF(OBJ@5,EQ,0,THEN,NEXT,ELSE,:ENDIF32)

  ;If the transporter is not working yet

  IF(@@RECQUE(0,0),NE,0,THEN,NEXT,ELSE,:ENDIF32)

    ;If there is any inorder waiting

    SV(@KK,1)
    :LOOP19
     SV(@LL,@@RECQUE(@KK,0))
     IF(@@NUPARE(@LL,1),NE,0,THEN,NEXT,ELSE,:ENDIF49)
       SV(OBJ@5,@LL)
       SV(@KK,@@RECQUE(0,0))
     :ENDIF49
     IV(@KK)
    IF(@KK,LE,@@RECQUE(0,0),THEN,:LOOP19)
:ENDIF32
IF(OBJ@5,NE,0,THEN,NEXT,ELSE,:ENDIF34)

  ;If the transporter has work to do

  SV(@KFLAG,1)
  IF(@@NUPARE(OBJ@5,1),NE,0,THEN,NEXT,ELSE,:ENDIF34)
   SV(@@ATREAT(OBJ@SN,1),@@NUPARE(OBJ@5,2))
    SV(@II,@@FREECE(0,0))
    DV(@@FREECE(0,0))
   SV(@@ATREAT(OBJ@SN,2),@@FREECE(@II,0))
    SV(@KFLAG,0)
:ENDIF34
IF(@KFLAG,EQ,0,THEN,NEXT,ELSE,:ELSE36)

  ;Pick the pallet in the reception and take it to the cell

  DV(@@NUPARE(OBJ@5,1))
  LK(!RETOCE)
  JP(:ENDIF36)
:ELSE36
  IF(OBJ@5,EQ,0,THEN,NEXT,ELSE,ELSE,:ELSE48)

    ;Returns to park

    SV(@PARFLA,1)
    IF(@@REATCO(OBJ@SN,1),NE,OBJ@1,THEN,NEXT,ELSE,:ENDIF41)
      SV(@PARFLA,0)
    :ENDIF41
    IF(@@REATCO(OBJ@SN,2),NE,OBJ@2,THEN,NEXT,ELSE,:ENDIF42)
      SV(@PARFLA,0)
```

```
:ENDIF42
IF(@PARFLA,EQ,0,THEN,NEXT,ELSE,:ENDIF43)

 ;Returns to park

 SV(OBJ@3,@@REATCO(OBJ@SN,1))
 SV(OBJ@4,@@REATCO(OBJ@SN,2))
  IF(OBJ@SN,EQ,1,THEN,NEXT,ELSE,:ELSE44)
    AO(OBJ@3,+,2)
    JP(:ENDIF44)
  :ELSE44
     AO(OBJ@3,-,2)
    :ENDIF44
    LK(!MOVE)
    LK(!REPA)
   :ENDIF43
   WE
   JP(:ENDIF48)
  :ELSE48
   SV(OBJ@5,0)
  :ENDIF48
 :ENDIF36
 JP(:RBEGIN)
ER

BL(!TESINO)
 IF(@@INOFLA(@PN,0),EQ,0,THEN,NEXT,ELSE,:ENDIF70)
  IF(@@NPROPA(@PN,0),LT,@@MINPRO(@PN,0),THEN,NEXT,ELSE,:ENDIF61)
   SV(@@INOFLA(@PN,0),1)
  IF(@@RECQUE(0,0),LT,3,THEN,NEXT,ELSE,:ELSE62)

   ;If there is a free reception point

   LK(!INOARR)
    SE
    JP(:ENDIF62)
  :ELSE62
   IV(@NINOWA)
   SV(@@NPROIN(@NINOWA,0),@PN)
   IF(@NINOWA,EQ,1,THEN,NEXT,ELSE,:ENDIF63)
     PM(XY(1,22),I.W.:     )
   :ENDIF63
   PV(XY(6,22),@NINOWA)
  :ENDIF62
 :ENDIF61
:ENDIF70
EL

;Outorder arrival

BL(!OUTARR)
  IV(@@DESQUE(0,0))
  SV(@IU,1)
  :LOOP7
   IF(@@DESFLA(@IU,0),EQ,0,THEN,NEXT,ELSE,:ENDIF26)
    SV(@@DESFLA(@IU,0),1)
    RV(N,@PALORD,@PALOME,@PALOSD)
    SV(@PP,0)
    SV(@IO,1)
    :LOOP8
```

```
SV(@@NUPADE(@IU,@IO),@@PROROT(@IO,0))
AO(@@NUPADE(@IU,@IO),*,@PALORD)
AO(@@NUPADE(@IU,@IO),/,100)
AO(@PP,+,@@NUPADE(@IU,@IO))
IV(@IO)
IF(@IO,LE,@NUMPRO,THEN,:LOOP8)
SV(@@NUPADE(@IU,0),@PP)
SV(@TMP,@@LOCDES(@IU,2))
AO(@TMP,-,1)
AO(@TMP,*,80)
AO(@TMP,*,2)
SV(@TMP1,@@LOCDES(@IU,1))
AO(@TMP1,+,2)
AO(@TMP1,*,2)
AO(@TMP,+,@TMP1)
SV(*DIPO,@TMP)
PV(*DIPO,@PP)
SV(@JC,@@DESQUE(0,0))
SV(@@DESQUE(@JC,0),@IU)
SV(@IU,3)
:ENDIF26
IV(@IU)
IF(@IU,LE,3,THEN,:LOOP7)
EL

;Inorder arrival

BL(!INOARR)
IV(@@RECQUE(0,0))
SV(@IU,1)
:LOOP17
IF(@@RECFLA(@IU,0),EQ,0,THEN,NEXT,ELSE,:ENDIF46)
SV(@@RECFLA(@IU,0),1)
SV(@PALORD,@@NUPARO(@PN,0))
SV(@@NUPARE(@IU,0),@PALORD)
SV(@@NUPARE(@IU,1),@PALORD)
SV(@@NUPARE(@IU,2),@PN)
SV(@TMP,@@LOCREC(@IU,2))
AO(@TMP,*,80)
AO(@TMP,*,2)
SV(@TMP1,@@LOCREC(@IU,1))
AO(@TMP1,-,6)
AO(@TMP1,*,2)
AO(@TMP,+,@TMP1)
SV(*DIPO,@TMP)
PV(*DIPO,@PALORD)
SV(@JC,@@RECQUE(0,0))
SV(@@RECQUE(@JC,0),@IU)
SV(@IU,3)
:ENDIF46
IV(@IU)
IF(@IU,LE,3,THEN,:LOOP17)
EL

;Move the pick transporter to the cell

BL(!PITOCE)

SV(@NC,@@ATPICT(OBJ@SN,2))
LK(!COORD)
```

```
SV(OBJ@3,@XE)                    ;Defines the xx destin coordinate
SV(OBJ@4,@YE)                    ;Defines the yy destin coordinate
SV(@TMP,@NE)
 AO(@TMP,/,2)
SV(@@FAC(OBJ@IN,0),@NE)
 AO(@TMP,*,2)
AO(@@FAC(OBJ@IN,0),-,@TMP)

;Correct the yy destin coordinate depending on the facing of the cell

 IF(@@FAC(OBJ@IN,0),EQ,0,THEN,NEXT,ELSE,:ELSE17)
  AO(OBJ@4,-,3)
  JP(:ENDIF17)
:ELSE17
  AO(OBJ@4,+,2)
:ENDIF17

;Move the transporter

LK(!MOVE)
IF(@@FAC(OBJ@IN,0),EQ,0,THEN,NEXT,ELSE,:ENDIF18)
   MD(1,1)
   AO(OBJ@2,+,1)
 :ENDIF18
 WT(@TIMPIC)                     ;Waits the picking delay
SV(@NC,@@ATPICT(OBJ@SN,2))
 SV(@KK,@@FREECE(0,0))
 IV(@KK)
  SV(@JJ,1)
:LOOP20
  SV(@LL,@@FREECE(@JJ,0))
  IF(@@PRICEL(@NC,0),LE,@@PRICEL(@LL,0),THEN,NEXT,ELSE,:ENDIF90)
    SV(@KK,@JJ)
    SV(@JJ,@@FREECE(0,0))
  :ENDIF90
  IV(@JJ)
 IF(@JJ,LE,@@FREECE(0,0),THEN,:LOOP20)
 SV(@JJ,@@FREECE(0,0))
 :WHILE1
 IF(@JJ,GE,@KK,THEN,NEXT,ELSE,:EWHILE1)
   SV(@LL,@JJ)
   IV(@LL)
   SV(@@FREECE(@LL,0),@@FREECE(@JJ,0))
   DV(@JJ)
   JP(:WHILE1)
 :EWHILE1
   IV(@@FREECE(0,0))
   SV(@@FREECE(@KK,0),@NC)
   SV(@PN,@@ATPICT(OBJ@SN,1))
   LK(!TESINO)
   LK(!COORD)
    PM(*CELL, )                  ;Erase the pallet of the cell
   LK(!CHOBJC)
   IF(@@FAC(OBJ@IN,0),NE,0,THEN,NEXT,ELSE,:ENDIF19)
     MD(1,1)
     AO(OBJ@2,+,1)
    :ENDIF19
    SV(OBJ@6,1)                  ;Position horizontal right
EL
BL(!CHOBJC)
```

```
   IF(@PN,NE,1,THEN,:KL1)
    SV(OBJ@ID,"A)
:KL1
   IF(@PN,NE,2,THEN,:KL2)
    SV(OBJ@ID,"B)
:KL2
   IF(@PN,NE,3,THEN,:KL3)
    SV(OBJ@ID,"C)
:KL3
   IF(@PN,NE,4,THEN,:KL4)
    SV(OBJ@ID,"D)
:KL4
EL

;Move the reach transporter to the reception

BL(!RETOCE)
   SV(OBJ@3,@@LOCREC(OBJ@5,1))
   SV(OBJ@4,@@LOCREC(OBJ@5,2))
   LK(!MOVE)
   WT(@TIMREC)
   SV(@PN,@@ATREAT(OBJ@SN,1))
   LK(!CHOBJC)
   DV(@@NUPARE(OBJ@5,0))
   SV(@TMP,@@LOCREC(OBJ@5,2))
   AO(@TMP,*,80)
   AO(@TMP,*,2)
   SV(@TMP1,@@LOCREC(OBJ@5,1))
   AO(@TMP1,-,6)
AO(@TMP1,*,2)
AO(@TMP,+,@TMP1)
SV(*DIPO,@TMP)
PV(*DIPO,@@NUPARE(OBJ@5,0))
IF(@@NUPARE(OBJ@5,0),EQ,0,THEN,NEXT,ELSE,:ENDIF40)

  ;If the inorder is finished

 PM(*DIPO,∞∞∞∞∞∞∞∞∞)
  SV(@@INOFLA(@PN,0),0)
  SV(@IQ,1)
  :LOOP15
   IF(OBJ@5,EQ,@@RECQUE(@IQ,0),THEN,NEXT,ELSE,:ENDIF45)
     SV(@JQ,@IQ)
    :ENDIF45
    IV(@IQ)
  IF(@IQ,LE,@@RECQUE(0,0),THEN,:LOOP15)
DV(@@RECQUE(0,0))
 :LOOP16
IF(@JQ,LE,@@RECQUE(0,0),THEN,NEXT,ELSE,:EXIT16)
  SV(@KQ,@JQ)
  IV(@KQ)
  SV(@@RECQUE(@JQ,0),@@RECQUE(@KQ,0))
  IV(@JQ)
  JP(:LOOP16)
 :EXIT16
 SV(@@RECFLA(OBJ@5,0),0)
 SV(OBJ@5,0)
 IF(@NINOWA,GT,0,THEN,NEXT,ELSE,:ENDIF50)
  SV(@PN,@@NPROIN(@NINOWA,0))
  DV(@NINOWA)
```

```
   IF(@NINOWA,EQ,0,THEN,NEXT,ELSE,:ELSE57)
   PM(XY(1,22),∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞∞)
    JP(:ENDIF57)
   :ELSE57
    PV(XY(6,22),@NINOWA)
   :ENDIF57
   LK(!INOARR)
  :ENDIF50
:ENDIF40
MD(1,1)
AO(OBJ@2,+,1)
SV(OBJ@6,1)
SV(@NC,@@ATREAT(OBJ@SN,2))
LK(!COORD)
SV(OBJ@3,@XE)                    ;Defines the xx destin coordinate
SV(OBJ@4,@YE)                    ;Defines the yy destin coordinate
SV(@TMP,@NE)
AO(@TMP,/,2)
SV(@@FAC(OBJ@IN,0),@NE)
AO(@TMP,*,2)
AO(@@FAC(OBJ@IN,0),-,@TMP)

;Correct the yy destin coordinate depending on the facing of the cell

IF(@@FAC(OBJ@IN,0),EQ,0,THEN,NEXT,ELSE,:ELSE37)
  AO(OBJ@4,-,2)
  JP(:ENDIF37)
 :ELSE37
  AO(OBJ@4,+,3)
:ENDIF37

;Move the transporter

LK(!MOVE)
IF(@@FAC(OBJ@IN,0),NE,0,THEN,NEXT,ELSE,:ENDIF38)
  MU(1,1)
  AO(OBJ@2,-,1)
 :ENDIF38
 WT(@TIMPIC)                     ;Waits the picking delay
SV(@NC,@@ATREAT(OBJ@SN,2))
SV(@PN,@@ATREAT(OBJ@SN,1))
SV(@JJ,@@NPROPA(@PN,0))
:WHILE2
  IF(@JJ,GT,0,THEN,NEXT,ELSE,:EWHILE2)
  SV(@LL,@JJ)
  IV(@LL)
  SV(@@NPROPA(@PN,@LL),@@NPROPA(@PN,@JJ))
  DV(@JJ)
  JP(:WHILE2)
:EWHILE2
IV(@@NPROPA(@PN,0))
SV(@@NPROPA(@PN,1),@NC)
LK(!COORD)
SV(OBJ@ID," )
LK(!PUPROC)
IF(@@FAC(OBJ@IN,0),EQ,0,THEN,NEXT,ELSE,:ENDIF39)
  MU(1,1)
  AO(OBJ@2,-,1)
 :ENDIF39
 SV(OBJ@6,2)                     ;Position horizontal left
```

```
EL
BL(!PUPROC)
      IF(@PN,NE,1,THEN,:L1)
        PM(*CELL,A)
:L1
      IF(@PN,NE,2,THEN,:L2)
        PM(*CELL,B)
:L2
      IF(@PN,NE,3,THEN,:L3)
        PM(*CELL,C)
:L3
      IF(@PN,NE,4,THEN,:L4)
        PM(*CELL,D)
:L4
EL

;Defines the link to calculate the cell coordinates

BL(!COORD)

   ;Calculates the racking number NE

   SV(@NE,@NC)
   AO(@NE,-,1)
   AO(@NE,/,22)
   AO(@NE,+,1)

   ;Calculates the racking cell number NCE

   SV(@TMP,@NE)
   AO(@TMP,-,1)
   AO(@TMP,*,22)
   SV(@NCE,@NC)
   AO(@NCE,-,@TMP)

   ;Calculates the cell yy coordinate

   SV(@YE,@NE)
   AO(@YE,/,2)
   SV(@TMP,@YE)
   AO(@YE,*,7)
   SV(@TMP1,@NE)
   AO(@TMP,*,2)
   AO(@TMP1,-,@TMP)
   AO(@TMP1,*,2)
   AO(@YE,+,@TMP1)

   ;Calculates the cell xx coordinate

   SV(@XE,@NCE)
   AO(@XE,-,1)
   AO(@XE,*,2)
   AO(@XE,+,18)
   SV(@TMP,@YE)
   AO(@TMP,*,80)
   AO(@TMP,*,2)
   SV(@TMP1,@XE)
   AO(@TMP1,*,2)
   AO(@TMP,+,@TMP1)
   SV(*CELL,@TMP)
```

```
EL

    ;Defines the link to put the pick transporters in the park

BL(!PIPA)

    ;Initialize the job attributes

  SV(OBJ@1,@@PICTCO(OBJ@SN,1))      ;Current location xx coordinate
  SV(OBJ@2,@@PICTCO(OBJ@SN,2))      ;Current location yy coordinate

    ;Position the transporter in the park

  SV(@TMP,OBJ@2)
  AO(@TMP,*,80)
  AO(@TMP,*,2)
  SV(@TMP1,OBJ@1)
  AO(@TMP1,*,2)
  AO(@TMP,+,@TMP1)
  SV(*TRAC,@TMP)
  MA(*TRAC,0)
EL

    ;Defines the link to put the reach transporters in the park

BL(!REPA)

    ;Initialize the job attributes

  SV(OBJ@1,@@REATCO(OBJ@SN,1))      ;Current location xx coordinate
  SV(OBJ@2,@@REATCO(OBJ@SN,2))      ;Current location yy coordinate

    ;Position the transporter in the park

  SV(@TMP,OBJ@2)
  AO(@TMP,*,80)
  AO(@TMP,*,2)
  SV(@TMP1,OBJ@1)
  AO(@TMP1,*,2)
  AO(@TMP,+,@TMP1)
  SV(*TRAC,@TMP)
  MA(*TRAC,0)
EL

;Defines the LINK to move a transporter

BL(!MOVE)
:AGAIN
IF(OBJ@6,LE,2,THEN,NEXT,ELSE,:ELSE1)
 IF(OBJ@2,EQ,OBJ@4,THEN,NEXT,ELSE,:ELSE2)
  IF(OBJ@1,GT,OBJ@3,THEN,NEXT,ELSE,:ELSE3)
   SV(@@DIF(OBJ@IN,0),OBJ@1)
   AO(@@DIF(OBJ@IN,0),-,OBJ@3)
   SV(@DIFF,@@DIF(OBJ@IN,0))
   ML(@DIFF,1)
   SV(OBJ@1,OBJ@3)
   SV(OBJ@6,2)
   JP(:ENDIF3)
  :ELSE3
   SV(@@DIF(OBJ@IN,0),OBJ@3)
```

```
  AO(@@DIF(OBJ@IN,0),-,OBJ@1)
  SV(@DIFF,@@DIF(OBJ@IN,0))
   MR(@DIFF,1)
  SV(OBJ@1,OBJ@3)
  SV(OBJ@6,1)
 :ENDIF3
 JP(:ENDIF2)
:ELSE2
 IF(OBJ@1,GT,OBJ@3,THEN,NEXT,ELSE,:ELSE4)
   IF(OBJ@1,GT,64,THEN,NEXT,ELSE,:ELSE5)
    SV(@@DIF(OBJ@IN,0),OBJ@1)
  AO(@@DIF(OBJ@IN,0),-,64)
  SV(OBJ@1,64)
  JP(:ENDIF5)
:ELSE5
 SV(@@DIF(OBJ@IN,0),OBJ@1)
 AO(@@DIF(OBJ@IN,0),-,14)
  SV(OBJ@1,14)
:ENDIF5
IF(OBJ@2,GT,OBJ@4,THEN,NEXT,ELSE,:ELSE6)
 DV(@@DIF(OBJ@IN,0))
  SV(OBJ@6,3)
  IV(OBJ@1)
   JP(:ENDIF6)
 :ELSE6
  IV(@@DIF(OBJ@IN,0))
   DV(OBJ@1)
   SV(OBJ@6,4)
 :ENDIF6
 SV(@DIFF,@@DIF(OBJ@IN,0))
  ML(@DIFF,1)
  JP(:ENDIF4)
:ELSE4
  IF(OBJ@1,LT,14,THEN,NEXT,ELSE,:ELSE7)
    SV(@@DIF(OBJ@IN,0),14)
    AO(@@DIF(OBJ@IN,0),-,OBJ@1)
     SV(OBJ@1,14)
     JP(:ENDIF7)
   :ELSE7
    SV(@@DIF(OBJ@IN,0),64)
    AO(@@DIF(OBJ@IN,0),-,OBJ@1)
     SV(OBJ@1,64)
   :ENDIF7
   IF(OBJ@2,GT,OBJ@4,THEN,NEXT,ELSE,:ELSE8)
    IV(@@DIF(OBJ@IN,0))
    IV(OBJ@1)
     SV(OBJ@6,3)
     JP(:ENDIF8)
    :ELSE8
     DV(@@DIF(OBJ@IN,0))
     DV(OBJ@1)
     SV(OBJ@6,4)
    :ENDIF8
    SV(@DIFF,@@DIF(OBJ@IN,0))
     MR(@DIFF,1)
  :ENDIF4
  JP(:ENDIF1)
 :ENDIF2
 JP(:ENDIF1)
:ELSE1
```

```
IF(OBJ@1,EQ,OBJ@3,THEN,NEXT,ELSE,:ELSE9)
  IF(OBJ@2,GT,OBJ@4,THEN,NEXT,ELSE,:ELSE10)
    SV(@@DIF(OBJ@IN,0),OBJ@2)
    AO(@@DIF(OBJ@IN,0),-,OBJ@4)
    SV(@DIFF,@@DIF(OBJ@IN,0))
     MU(@DIFF,1)
     SV(OBJ@2,OBJ@4)
     SV(OBJ@6,3)
     JP(:ENDIF9)
   :ELSE10
     SV(@@DIF(OBJ@IN,0),OBJ@4)
     AO(@@DIF(OBJ@IN,0),-,OBJ@2)
     SV(@DIFF,@@DIF(OBJ@IN,0))
      MD(@DIFF,1)
      SV(OBJ@2,OBJ@4)
      SV(OBJ@6,4)
  JP(:ENDIF9)
:ELSE9
  IF(OBJ@2,GT,OBJ@4,THEN,NEXT,ELSE,:ELSE11)
    SV(@@DIF(OBJ@IN,0),OBJ@2)
    AO(@@DIF(OBJ@IN,0),-,OBJ@4)
    SV(@DIFF,@@DIF(OBJ@IN,0))
     MU(@DIFF,1)
     SV(OBJ@6,1)
     SV(OBJ@2,OBJ@4)
     JP(:ENDIF9)
   :ELSE11
     SV(@@DIF(OBJ@IN,0),OBJ@4)
     AO(@@DIF(OBJ@IN,0),-,OBJ@2)
     SV(@DIFF,@@DIF(OBJ@IN,0))
      MD(@DIFF,1)
      SV(OBJ@6,1)
    SV(OBJ@2,OBJ@4)
  :ENDIF9
:ENDIF1
IF(OBJ@1,NE,OBJ@3,THEN,:AGAIN)
IF(OBJ@2,NE,OBJ@4,THEN,:AGAIN)
EL
```

APPENDIX V.A

STDIO Subroutines Library Description

INITIALIZATION

Subroutine OPEN ()

- Open, if required, a binary I/O channel associated with the terminal, bypassing the operating system so that all characters (including control characters) can be used.

Subroutine CONFI ()

- Set the system specific configuration and reset the parameters to the initial conditions.

CURSOR CONTROL

Subroutine SETAB (NPOSI)

- Move the cursor NPOSI positions down.

  NPOSI: Number of cursor positions down (integer*4)

Subroutine SETAC (NPOSI)

- Move the cursor NPOSI positions up.

  NPOSI: Number of cursor positions up (integer*4)

Subroutine SETAD (NPOSI)

- Move the cursor NPOSI positions to the right.

  NPOSI: Number of cursor positions to the right (integer*4)

Subroutine SETAE (NPOSI)

- Move the cursor NPOSI positions to the left.

  NPOSI: Number of cursor positions to the left (integer*4)

Subroutine WRICA (NLINE, NCOLU)

- Move the cursor to the NLINE, NCOLU position on the screen.

  NLINE: Screen line number (integer*4)

  NCOLU: Screen column number (integer*4)

CHARACTER INPUT

Character*1 Function GETCH ()

- Read a character from the keyboard (terminal).

CHARACTER OUTPUT

Subroutine PUTCH (CHARS)

- Send a string of characters to the screen (terminal).

  CHARS: Characters to be sended to the screen (character*(*))

Subroutine WRICH (NLINE, NCOLU, CHARS)

- Writes a string of characters at NLINE, NCOLU on the screen.

  NLINE: Screen line number (integer*4)

  NCOLU: Screen column number (integer*4)

  CHARS: Characters to be written on the screen (character*(*))

CHARACTER ATTRIBUTES

Subroutine DEFBL (BLINK)

- Set the blink attribute ON/OFF.

  BLINK: Blink attribute ("Y" - ON; "N" - OFF) (character*1)

Subroutine DEFCA (UNDER, BLINK, DOUBL)

- Set the underline, blink and double size character attributes.

  UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)

  BLINK: Blink attribute ("Y" - ON; "N" - OFF) (character*1)

  DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)

Subroutine DEFCB (NBACO)

- Defines the character background colour.

  NBACO: Character background colour index (integer*4)

Subroutine DEFCF (NFOCO)

- Defines the character foreground colour.

  NFOCO: Character foreground colour index (integer*4)

Subroutine DEFCL (NERCO)

- Defines the screen erase colour.

NERCO: Screen erase colour index (integer*4)

Subroutine DEFCO (NBACO, NFOCO, NERCO)

- Defines the character background, foreground and screen erase colours.

NBACO: Character background colour index (integer*4)
NFOCO: Character foreground colour index (integer*4)
NERCO: Screen erase colour index (integer*4)

Subroutine DEFDO (DOUBL)

- Set the double size character attribute ON/OFF.

DOUBL: Doubl attribute ("Y" - ON; "N" - OFF) (character*1)

Subroutine DEFGL (NFOCO, NBACO, UNDER, DOUBL)

- Defines the foreground, background, underline and double size attributes.

NFOCO: Character foreground colour index (integer*4)
NBACO: Character background colour index (integer*4)
UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)
DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)

Subroutine DEFUN (UNDER)

- Set the underline character attribute ON/OFF.

UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)

MESSAGES

Subroutine MENUS (TEXTM)

- Writes a message at the bottom of the screen using the default character attributes.

TEXTM: Text to be written (character*(*))

Subroutine MERRO (ERROM)

- Writes an error message at the bottom of the screen using the default error messages character attributes.

ERROM: Error message to be written (character*(*))

Subroutine BELLE ()

- Sounds the terminal bell.

SCREEN ERASE

Subroutine EREOL ()

- Erases the screen from the current cursor position to the end of the line.

Subroutine ERPAG ()

- Erases the entire screen.

Subroutine LIMPA (NINLI, NFILI, NINCO, NFICO, NBACO)

- Erases part of the screen with a specific colour.

NINLI: Initial line (integer*4)

NFILI: Final line (integer*4)

NINCO: Initial column (integer*4)

NFICO: Final column (integer*4)

NBACO: Background colour index (integer*4)

APPENDIX V.B

TEK Subroutines Library Description

Subroutine BEGINPANEL (NCORX, NCORY, NCOFR)

- Starts a panel definition.

NCORX: First point xx coordinate (integer*4)

NCORY: First point yy coordinate (integer*4)

NCOFR: Draw-boundary mode (integer*4)

Subroutine BORDER (NCONT)

- Controls the visibility of a border drawn around the current view's viewport.

NCONT: Border-visibility mode (integer*4)

Subroutine BYPASSCANCEL (NCARE)

- Specifies the character that causes the terminal to leave Bypass mode.

NCARE: Bypass cancel character (integer*4)

Subroutine CLEARDIALOG

- Clears (erases) the dialog buffer.

Character*3 Function CODEI (INTEI)

- Returns a string representing the integer in the Tektronix format.

INTEI: Integer value (integer*4)

Subroutine COREP (IX, IY, ICARE)

- Returns the graphic coordinates and the character of theTektronix GIN report.

IX, IY: Graphic coordinates (integer*4)

ICARE: ASCII code from the key pressed (integer*4)

Character*5 Function CORGR (IX, IY)

- Returns the Tektronix character codes for the graphic coordinates IX and IY.

IX, IY: Graphic coordinates (integer*4)

Subroutine DABUFFER (NLINE)

- Specifies the maximum number of lines of text stored in the dialog area buffer.

NLINE: Number of lines (integer*4)

Subroutine DAINDEX (NCORC, NCORB, NCORL)

- Specifies the colour indices for alphatext and backgrounds.

NCORC: Character index (integer*4)

NCORB: Character background index (integer*4)

NCORC: Dialog background index (integer*4)

Subroutine DALINES (NLINE)

- Specifies the maximum number of lines visible in the dialog area.

NLINE: Number of lines (integer*4)

Subroutine DAVISIBILITY (NCONT)

- Specifies whether the dialog area is visibled.

NCONT: Visibility mode (integer*4)

Subroutine DEFINE (NUMAC, NASCI, NDIME)

- Creates and deletes macros, which can be expanded on command from the host, at the keyboard, or both.

NUMAC: Macro number (integer*4)

NASCI: Array with the characters codes definning the macro

(integer*4 (NDIME))

NDIME: NASCI array maximum dimension (integer*4)

Subroutine DRAW (NCORX, NCORY)

- Draws a vector from the current graphics position to a specified location.

NCORX, NCORY: Graphic coordinates (integer*4)

Subroutine EDITMARGIN (NTOPM, NBOTM)

- Sets top and bottom margins to define scrolling region with fixed regions above and below.

NTOPM, NBOTM: Top and bottom margins (integer*4)

Subroutine ENDPANEL

- Terminates a panel definition.

Subroutine EOLSTRING (NDIME, NVECT)

- Specifies the terminal's EOF string.

NDIME, NVECT: Number of codes and ASCII codes array (integer*4)

Subroutine ESCAL (...)

- Set the scale for writing text in graphics mode.

Subroutine ESCRG (...)

- Writes a text string in graphics mode.

Subroutine ESING (...)

- Writes an integer value in graphics mode.

Subroutine ESREG (...)

- Writes a floating point value in graphics mode.

Subroutine FILLPATTERN (NCORE)

- Specifies the fill pattern for subsequent panels.

NCORE: Fill pattern number (integer*4)

Subroutine FIXUP (NCONT)

- Specifies when the screen display is updated after changes have been made.

NCONT: Fixup level parameter (integer*4)

Subroutine GCURSOR (NCOR1, NCOR2, NCOR3)

- Defines the colour mixture for the graphics cursor.

NCOR1, NCOR2, NCOR3: Colour coordinates (integer*4)

Subroutine GINCURSOR (NCONT, NSEGM)

- Selects a segment for use as the cursor.

NCONT: Device function code (integer*4)

NSEGM: Segment number (integer*4)

Subroutine GINDISABLE (NCONT)

- Disables the GIN function.

NCONT: Device function code (integer*4)

Subroutine GINENABLE (NCONT, NUMGI)

- Enables the GIN function for a number of events.

NCONT: Device function code (integer*4)

NUMGI: Number of GIN events (integer*4)

Subroutine GINGRIDDING (NCONT, NESPX, NESPY)

- Restricts the GIN positions to the points of a grid.

NCONT: Device function code (integer*4)

NESPX, NESPY: Grid space along x and y (integer*4)

Subroutine GINREP (NCONT, NCORX, NCORY, NCARE)

- Forces the terminal to send a GIN report.

NCONT: Device function code (integer*4)

NCORX, NCORY: GIN cursor coordinates (integer*4)

NCARE: ASCII code if a key was pressed (integer*4)

Subroutine GINRUBBER (NCONT, NMODE)

- Turns the rubberbanding facility ON or OFF.

NCONT: Device function code (integer*4)

NMODE: Rubberbanding mode parameter (integer*4)

Subroutine GTEXT (ALFAN)

- Writes a text string at the current graphics position.

ALFAN: Text (character*(*))

Subroutine GTINDEX (NCORE)

- Specifies the colour index for text in the graphics area.

NCORE: Text colour index (integer*4)

Subroutine GTROTATION (ANGUL)

- Specifies the rotation angle for writing graphics text.

ANGUL: Text rotation angle (real*4)

Subroutine GTSIZE (NLARG, NALTU, NESPA)

- Specifies the graphics text size.

NLARG, NALTU, NESPA: Width, height and spacing (integer*4)

Subroutine INITG

- Initializes the graphics terminal.

Character*2 Function INTCA (INTEI)

- Transforms an integer value into tektronix character codes.

INTEI: Integer value (integer*4)

Subroutine LECLG (...)

- Reads a character string with editting facilities.

Subroutine LEING (...)

- Reads and validate an integer value.

Subroutine LEREG (...)

- Reads and validate a floating point value.

Subroutine LESCA (...)

- Returns the current settings for writing graphics text.

Subroutine LINEI (NCORE)

- Specifies the colour index for drawing lines.

NCORE: Line colour index (integer*4)

Subroutine LINES (NTLIN)

- Specifies the line style for drawing lines.

NTLIN: Line style (integer*4)

Subroutine MARKER (NCORX, NCORY)

- Draws a marker at the specified location.

NCORX, NCORY: Graphic coordinates (integer*4)

Subroutine MARKERTYPE (NMARK)

- Defines the marker type to be drawn.

NMARK: Marker type (integer*4)

Subroutine MOVE (NCORX, NCORY)

- Sets the current graphics position.

NCORX, NCORY: Graphic coordinates (integer*4)

Subroutine PAGE

- Erases the graphics screen.

Subroutine PXBEGIN (NSURF, MOALU, NBIPI)

- Sets the parameters to be used in pixel operations.

NSURF: Surface number (integer*4)

MOALU: Arithmetic Logic Unit (ALU) mode (integer*4)

NBIPI: Number of bits per pixel (integer*4)

Subroutine PXRECTANGLE (NCOX1, NCOY1, NCOX2, NCOY2, NINCO)

- Sets the parameters to be used in pixel operations.

NCOX1, NCOY1: Lower left corner xx and yy coordinates (integer*4)

NCOX2, NCOY3: Upper right corner xx and yy coordinates (integer*4)

NINCO: Colour index to fill the rectangle (integer*4)

Subroutine QUEUESIZE (NSIZE)

- Specifies the size of the terminal's input queue.

NSIZE: Number of bytes of the terminal's input queue (integer*4)

Character*6 Function REALC (REALT)

- Transforms a floating point value into tektronix character codes.

REALT: Floating point value (REAL*4)

Subroutine REOM (NEOMF)

- Specifies how often the terminal sends EOM indicators to the host.

NEOMF: End Of Message (EOM) frequency parameter (integer*4)

Subroutine RM (NCONT)

- Resets one or more terminal modes.

NCONT: Reset mode parameter (integer*4)

Subroutine RSIGCHARS (NREPO, NRESG, NTESG)

- Sets the signature characters used in reports that the terminal sends to the host.

  NREPO: Report type code (integer*4)

  NRESG: ASCII code of the report signature character (integer*4)

  NTESG: ASCII code of the terminal signature character (integer*4)

Subroutine SDEFINITIONS (NSUPE, NPLAN)

- Specifies the number of bit planes in each surface.

  NSUPE: Number of surfaces to be defined (integer*4)

  NPLAN: Array with the number of bit planes for each surface (integer*4)

Subroutine SEL_CODE (MODE)

- Selects the terminal mode.

  MODE: Terminal mode (ANSI, TEK, VT-52) (integer*4)

Subroutine SGCLOSE

- Ends a segment definition.

Subroutine SGDELETE (NSEGM)

- Deletes the specified segment.

  NSEGM: Segment number (integer*4)

Subroutine SGHIGHLIGHT (NSEGM, NPISC)

- Turns highlighting on or off for the specified segments.

  NSEGM: Segment number (integer*4)

  NPISC: Highlighting mode (integer*4)

Subroutine SGMODE (NSEGM, NMOES)

- Selects the writing mode for displaying segments.

  NSEGM: Segment number (integer*4)

  NMOES: Writing mode (integer*4)

Subroutine SGOPEN (NSEGM)

- Begins the definition of a new segment.

  NSEGM: Segment number (integer*4)

Subroutine SGPIVOT (NCORX, NCORY)

- Sets the pivot point for segment definitions.

NCORX, NCORY: Pivot point xx and yy coordinates (integer*4)

Subroutine SGPOSITION (NSEGM, NCORX, NCORY)

- Moves a segment to the specified position.

NSEGM: Segment number (integer*4)

NCORX, NCORY: Segment pivot point xx and yy coordinates (integer*4)

Subroutine SGTRANSFORM (NSEGM, ESCAX, ESCAY, ANGUL, NCORX, NCORY)

- Scales, rotates and moves the specified segment.

NSEGM: Segment number (integer*4)

ESCAX, ESCAY: xx and yy scale factors (real*4)

ALGUL: Rotation angle (real*4)

NCORX, NCORY: Segment pivot point xx and yy coordinates (integer*4)

Subroutine SGVISIBILITY (NSEGM, NVISI)

- Sets the segment visibility on or off.

NSEGM: Segment number (integer*4)

NVISI: Visibility (integer*4)

Subroutine VATTRIBUTES (NSUPE, NCOAP, NCOFR)

- Sets the surface attributes.

NSUPE: Surface number (integer*4)

NCOAP: Wipe colour index (integer*4)

NCOFR: Border colour index (integer*4)

Subroutine VDELETE (NCONT)

- Deletes the specified view.

NCONT: View number (integer*4)

Subroutine VIEWPORT (NCOX1, NCOY1, NCOX2, NCOY2)

- Sets the position of the current view's viewport.

NCOX1, NCOY1: Lower left corner xx and yy coordinates (integer*4)

NCOX2, NCOY3: Upper right corner xx and yy coordinates (integer*4)

Subroutine VSELECT (NCONT)

- Selects the specified view.

NCONT: View number (integer*4)

Subroutine WINDOW (NCOX1, NCOY1, NCOX2, NCOY2)

- Sets the boundaries of the current view's window.

NCOX1, NCOY1: Lower left corner xx and yy coordinates (integer*4)

NCOX2, NCOY3: Upper right corner xx and yy coordinates (integer*4)

APPENDIX V.C

INSTA Subroutines Library Description

HIGH LEVEL WRITE

Subroutine ESCRI (TEXTM, NCOLU, NLINE, NBACO, NFOCO, BLINK, BELLS, DOUBL, UNDER)

- Writes text at the NLINE, NCOLU position on the screen with attributes.

TEXTM: Text to be written (character*(*))

NCOLU: Screen column number (integer*4)

NLINE: Screen line number (integer*4)

NBACO: Character background colour index (integer*4)

NFOCO: Character foreground colour index (integer*4)

BLINK: Blink attribute ("Y" - ON; "N" - OFF) (character*1)

BELLS: Sounds the bell ("Y" - YES; "N" - NO) (character*1)

DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)

UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)

Subroutine ESINC (INTEI, ALFAN, NLENG, IERRO)

- Writes an integer value to the ALFAN character variable.

INTEI: Integer value to be written (integer*4)

ALFAN: Character variable (character*(*))

NLENG: Field length (integer*4)

Input: Maximum field length

Output: Current field length

IERRO: Control variable (integer*4)

Input: Justify (=0 left; =1 right; =2 right pad with zeros)

Output: Error value (<>1 no error; =1 error)

Subroutine ESINT (INTEI, NLENG, NCOLU, NLINE, NBACO, NFOCO, DOUBL, UNDER, IERRO)

- Writes an integer value at NLINE, NCOLU on the screen with attributes.

INTEI: Integer value to be written (integer*4)

NLENG: Field length (integer*4)

NCOLU: Screen column number (integer*4)

NLINE: Screen line number (integer*4)

NBACO: Character background colour index (integer*4)
NFOCO: Character foreground colour index (integer*4)
DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)
UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)
IERRO: Control variable (integer*4)
Input: Justify (=0 left; =1 right; =2 right pad with zeros)
Output: Error value (<>1 no error; =1 error)

Subroutine ESREA (REALV, NLENG, NCOLU, NLINE, NBACO, NFOCO, DOUBL, UNDER, IERRO)

- Writes a real value at NLINE, NCOLU on the screen with attributes.

REALV: Real value to be written (real*4)
NLENG: Field length (integer*4)
NCOLU: Screen column number (integer*4)
NLINE: Screen line number (integer*4)
NBACO: Character background colour index (integer*4)
NFOCO: Character foreground colour index (integer*4)
DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)
UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)
IERRO: Control variable (integer*4)
Input: Not applicable
Output: Error value (<>1 no error; =1 error)

Subroutine ESREC (REALV, ALFAN, NLENG, IERRO)

- Writes a real value to the ALFAN character variable.

REALV: Real value to be written (real*4)
ALFAN: Character variable (character*(*))
NLENG: Field length (integer*4)
Input: Maximum field length
Output: Current field length
IERRO: Control variable (integer*4)
Input: not applicable
Output: Error value (<>1 no error; =1 error)

HIGH LEVEL READ

Subroutine LECAR (TEXTM, NLENG, NCOLU, NLINE, FMCHA, NMFOC, NBACO, NFOCO, DOUBL, UNDER, IERRO)

- Reads text at the NLINE, NCOLU position on the screen with attributes and field validation.

TEXTM: Returns the text read (character*(*))
NLENG: Field length (integer*4)
NCOLU: Screen column number (integer*4)
NLINE: Screen line number (integer*4)
FMCHA: Field mask character (character*1)
NMFOC: Field mask character foreground colour (integer*4)
NBACO: Character background colour index (integer*4)
NFOCO: Character foreground colour index (integer*4)
DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)
UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)
IERRO: Control variable (integer*4)
Input: nº of characters to fill (=0 all; =1 not all)
Output: Escape code character

Subroutine LECLI (TEXTM, NUCON, NASCI, NASCF, NLINE, NCOLU, NBACO, NFOCO, NERCO, DOUBL, UNDER, NEDIT, NCHAC, ICHAC, IERRO)

- Reads text at the NLINE, NCOLU position on the screen with attributes and full editing facilities.

TEXTM: Returns the text read (character*(*))
NUCON: Number of invalids ASCII code sets (integer*4)
NASCI: Vector with the sets initial ASCII codes (integer*4)
NASCF: Vector with the sets final ASCII codes (integer*4)
NLINE: Screen line number (integer*4)
NCOLU: Screen column number (integer*4)
NBACO: Character background colour index (integer*4)
NFOCO: Character foreground colour index (integer*4)
NERCO: Erase colour index (integer*4)
DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)
UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)
NEDIT: Edit control (integer*4)
Input: (=0 new; <>0 modify)
Output: Current number of characters

NCHAC: Number of escape code characters (integer*4)

ICHAC: Vector with the ASCII escape code characters (integer*4)

IERRO: Control variable (integer*4)

Input: not applicable

Output: Escape code character

Subroutine LEINT (INTEI, MIVAL, MAVAL, NLENG, NCOLU, NLINE, FMCHA, NMFOC, NBACO, NFOCO, DOUBL, UNDER, IERRO)

- Reads an integer value at the NLINE, NCOLU position on the screen with attributes and field validation.

INTEI: Returns the integer value read (integer*4)

MIVAL: Lower limit for the integer value (integer*4)

MAVAL: Upper limit for the integer value (integer*4)

NLENG: Field length (integer*4)

NCOLU: Screen column number (integer*4)

NLINE: Screen line number (integer*4)

FMCHA: Field mask character (character*1)

NMFOC: Field mask character foreground colour (integer*4)

NBACO: Character background colour index (integer*4)

NFOCO: Character foreground colour index (integer*4)

DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)

UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)

IERRO: Control variable (integer*4)

Input: nº of characters to fill (=0 all; =1 not all)

Output: Escape code character

Subroutine LEREA (REALV, MIVAL, MAVAL, NLENG, NCOLU, NLINE, FMCHA, NMFOC, NBACO, NFOCO, DOUBL, UNDER, IERRO)

- Reads a real value at the NLINE, NCOLU position on the screen with attributes and field validation.

REALV: Returns the real value read (real*4)

MIVAL: Lower limit for the real value (real*4)

MAVAL: Upper limit for the real value (real*4)

NLENG: Field length (integer*4)

NCOLU: Screen column number (integer*4)

NLINE: Screen line number (integer*4)

FMCHA: Field mask character (character*1)

NMFOC: Field mask character foreground colour (integer*4)

NBACO: Character background colour index (integer*4)

NFOCO: Character foreground colour index (integer*4)

DOUBL: Double size attribute ("Y" - ON; "N" - OFF) (character*1)

UNDER: Underline attribute ("Y" - ON; "N" - OFF) (character*1)

IERRO: Control variable (integer*4)

Input: nº of characters to fill (=0 all; =1 not all)

Output: Escape code character

SCREEN MANAGEMENT

Subroutine LEDEV (...)

- Reads all the information associated with a data screen created with the MSC data-entry screen generator (as it is an auxiliary subroutine and has a long parameter list the parameters description section was omited.

Subroutine LEFOVC (...)

- Controls a data screen created with the MSC data-entry screen generator (as it is an auxiliary subroutine and has a long parameter list the parameters description section was omited.

Subroutine LERMA (FILNA, TEXTV, INTEV, REALV, MODIF, IERRO)

- Reads a data screen created with the MSC data-entry screen generator with a fixed number of fields.

FILNA: Data screen file name (character*(*))

TEXTV: Vector with the character field values (character*(*))

INTEV: Vector with the integer field values (integer*4)

REALV: Vector with the real field values (real*4)

MODIF: Control variable (integer*4)

=0; displays and read the data screen

=1; displays and modify the data screen

=2; display the data screen and returns

=3; display only the field values and returns

=4; read without displaying the data screen

=5; modifies without displaying the data screen

IERRO: Control variable (integer*4)

Input: not applicable

Output: Escape code character

Subroutine LERMV (FILNA, TEXTV, INTEV, REALV, MODIF, NREPE, IERRO)

- Reads a data screen created with the MSC data-entry screen generator with a variable number of fields.

FILNA: Data screen file name (character*(*))

TEXTV: Vector with the character field values (character*(*))

INTEV: Vector with the integer field values (integer*4)

REALV: Vector with the real field values (real*4)

MODIF: Control variable (integer*4)

=0; displays and read the data screen

=1; displays and modify the data screen

=2; display the data screen and returns

=3; display only the field values and returns

=4; read without displaying the data screen

=5; modifies without displaying the data screen

NREPE: Field repetition number vector (integer*4)

IERRO: Control variable (integer*4)

Input: not applicable

Output: Escape code character

AUXILIARY

Function LENGH (ALFAN)

- Returns the number of characters in ALFAN.

ALFAN: Character string (character*(*))

LENGH: Number of characters in ALFAN (integer*4)

APPENDIX V.D

SIMVIS Subroutines Library Description

AUXILIARY

Subroutine EDELAY ()

- Delays the program execution for a fixed time.

Subroutine INFOZ0 ()

- Executes the INFOZ0 interaction which let the user inquire about the simulation model parameters.

Subroutine NOMPOI (IPOINT, NAMELE)

- Retrieves the element name given the element pointer.

IPOINT: Element pointer (I) (integer*4)

NAMELE: Element name (O) (character*(*))

Subroutine VERNOM (NUTYPE, NAMELE, ERRORM, IPOINT)

- Retrieves a pointer given the element name.

NUTYPE: Element type (I) (integer*4)

NAMELE: Element name (I) (character*(*))

ERRORM: Error message (O) (character*6)

IPOINT: Element pointer (O) (integer*4)

Subroutine WRITZ0 ()

- Writes to a file the internal simulation parameters.

DUMP MANAGEMENT

Subroutine DUMP00 (DUMPNA)

- Creates an initial dump.

DUMPNA: Dump file name (I) (character*(*))

Subroutine DUMPII (DUMPNA)

- Creates an intermediate dump.

DUMPNA: Dump file name (I) (character*(*))

Subroutine REST00 (DUMPNA)

- Restores an initial dump.

DUMPNA: Dump file name (I) (character*(*))

Subroutine RESTII (DUMPNA)

- Restores an intermediate dump.

DUMPNA: Dump file name (I) (character*(*))

ENTITY MANAGEMENT

Subroutine CRIENT (NOMENT, NUATNU, NUATAL, ICOREN, IENTPO)

- Creates an entity.

NOMENT: Entity name (I) (character*6)

NUATNU: Number of numeric attributes (I) (integer*4)

NUATAL: Number of alphanumeric attributes (I) (integer*4)

ICOREN: Colour code (I) (integer*4)

IENTPO: Entity pointer (O) (integer*4)

Subroutine DALENP (IENTPO, NUATRI, VATRIB)

- Defines an entity alphanumeric attribute.

IENTPO: Entity pointer (I) (integer*4)

NUATRI: Attribute number (I) (integer*4)

VATRIB: Attribute value (I) (character*(*))

Subroutine DANENP (IENTPO, NUATRI, VATRIB)

- Defines an entity numeric attribute.

IENTPO: Entity pointer (I) (integer*4)

NUATRI: Attribute number (I) (integer*4)

VATRIB: Attribute value (I) (real*4)

Subroutine LALENP (IENTPO, NUATRI, VATRIB)

- Reads an entity alphanumeric attribute.

IENTPO: Entity pointer (I) (integer*4)

NUATRI: Attribute number (I) (integer*4)

VATRIB: Attribute value (O) (character*(*))

Subroutine LANENP (IENTPO, NUATRI, VATRIB)

- Reads an entity numeric attribute.

IENTPO: Entity pointer (I) (integer*4)

NUATRI: Attribute number (I) (integer*4)

VATRIB: Attribute value (O) (real*4)

Subroutine LCANEN (IENTPO, NATRIB, NUATRI, VATRIB)

- Reads a set of entity numeric attributes.

IENTPO: Entity pointer (I) (integer*4)

NATRIB: Number of attributes to be read (I) (integer*4)

NUATRI: Number of the first attribute to be read (I) (integer*4)

VATRIB: Attribute value vector (O) (real*4)

Function READAT (IENTPO, NUATRI)

- Retrieves an entity numeric attribute.

IENTPO: Entity pointer (I) (integer*4)

NUATRI: Attribute number (I) (integer*4)

READAT: Attribute value (O) (real*4)

ERROR MANAGEMENT

Subroutine SERROR (NSUBRO, NERROR, ERRORS)

- Send an internal simulation error message.

NSUBRO: Subroutine number (I) (integer*4)

NERROR: Error number (I) (integer*4)

ERRORS: User application message (I) (character*(*))

EVENT MANAGEMENT

Subroutine ADENEV (IPOENT)

- Adds an entity to the system clock queue (scheduled events).

  IPOENT: Pointer to the entity (I) (integer*4)

Subroutine CRCLOC (MAXELE)

- Creates the system clock.

  MAXELE: Maximum number of elements in the clock (I) (integer*4)

Function IDENTI ()

- Retrieves the pointer to the entity involved with the current event.

  IDENTI: Entity pointer (O) (integer*4)

Subroutine NEXTEN ()

- Takes the next entity from the system clock queue.

Subroutine SCHEDU (IPOENT, ITIMES, NEVENT)

- Schedules a new event.

  IPOENT: Entity pointer (I) (integer*4)

  ITIMES: Event time (I) (integer*4)

  NEVENT: Event number (I) (integer*4)

Subroutine SYSEVE (NACTEV)

- Executes the system event number NACTEV.

  NACTEV: System event number (I) (integer*4)

Subroutine TAENEV (IPOENT)

- Takes an entity from the system clock queue (scheduled events).

  IPOENT: Entity pointer (I) (integer*4)

Subroutine UNSCHE (IPOENT)

- Unschedules an event.

  IPOENT: Entity pointer (I) (integer*4)

Subroutine USEEVE (NACTEV)

- Executes the user event number NACTEV.

NACTEV: User event number (I) (integer*4)

INITIALIZATION

Subroutine INITZ0 (KSIMTI, KCHENT, KCHQUE, KCHGRO, KCHECR, KUSINI, KUSINT, KUSEND, KAXELE, KLIBOX, KBABOX, KFOBOX, KLEBOX, LSIMTI, KUSEVE, KTIDEL)

- Initializes some of the simulation internal parameters.

KSIMTI: Simulation time (I) (interger*4)
KCHENT: Number of characters on an entity name (I) (integer*4)
KCHQUE: Number of characters on an queue name (I) (integer*4)
KCHGRO: Number of characters on an group name (I) (integer*4)
KCHECR: Number of characters on an display name (I) (integer*4)
KUSINI: Number of initial user interactions (I) (integer*4)
KUSINT: Number of intermediate user interactions (I) (integer*4)
KUSEND: Number of final user interactions (I) (integer*4)
KAXELE: Maximum number of elements in the clock (I) (integer*4)
KLIBOX: Interaction box initial line (I) (integer*4)
KBABOX: Interaction box background colour (I) (integer*4)
KFOBOX: Interaction box foreground colour (I) (integer*4)
KLEBOX: Interaction box last line (I) (integer*4)
LSIMTI: Maximum simulation time (I) (integer*4)
KUSEVE: Number of user events (I) (integer*4)
KTIDEL: Model execution delay (milis) (I) (integer*4)

INTERACTIONS

Subroutine INTSYS (NUINTE)

- Executes the system interaction number NUINTE.

NUINTE: System interaction number (I) (integer*4)

LOGICAL DISPLAY MANAGEMENT

Subroutine CRIECR (NUMECR, NUATNU, NUATAL, IONOFF, IECRAP)

- Creates a logical display.

NUMECR: Logical display number (I) (integer*4)
NUATNU: Number of numeric attributes (I) (integer*4)
NUATAL: Number of alphanumeric attributes (I) (integer*4)
IONOFF: ON/OFF display indicator (I) (integer*4)
IECRAP: Logical display pointer (O) (integer*4)

Subroutine DANECR (NUMECR, NUATRI, VATRIB)

- Defines a logical display numeric attribute.

NUMECR: Logical display number (I) (integer*4)
NUATRI: Attribute number (I) (integer*4)
VATRIB: Attribute value (I) (real*4)

Function INFECR (NUMECR)

- Retrieves the logical display status flag.

NUMECR: Logical display number (I) (integer*4)
INFECR: Status flag (=0 OFF; =1 ON) (O) (integer*4)

Subroutine LANECR (NUMECR, NUATRI, VATRIB)

- Reads a logical display numeric attribute.

NUMECR: Logical display number (I) (integer*4)
NUATRI: Attribute number (I) (integer*4)
VATRIB: Attribute value (O) (real*4)

Subroutine LCAECR (NUMECR, NATRIB, NUATRI, VATRIB)

- Reads a set of logical display numeric attributes.

NUMECR: Logical display number (I) (integer*4)
NATRIB: Number of attributes to be read (I) (integer*4)
NUATRI: Number of the first attribute to be read (I) (integer*4)
VATRIB: Attribute value vector (O) (real*4)

Function READIS (NUMECR, NUATRI)

- Retrieves a logical display numeric attribute.

NUMECR: Logical display number (I) (integer*4)

NUATRI: Attribute number (I) (integer*4)

READIS: Attribute value (O) (real*4)

Subroutine SCREOF (NUMECR)

- Turn the logical display OFF.

NUMECR: Logical display number (I) (integer*4)

Subroutine SCREON (NUMECR)

- Turn the logical display ON.

NUMECR: Logical display number (I) (integer*4)

QUEUE MANAGEMENT

Subroutine ADDFIR (IPOENT, IPOQUE)

- Adds an entity to the first position in a queue.

IPOENT: Pointer to the entity (I) (integer*4)

IPOQUE: Pointer to the queue (I) (integer*4)

Subroutine ADDLAS (IPOENT, IPOQUE)

- Adds an entity to the last position in a queue.

IPOENT: Pointer to the entity (I) (integer*4)

IPOQUE: Pointer to the queue (I) (integer*4)

Subroutine ADSQUE (IPOENT, IPOQUE, IATRIB)

- Adds an entity to a sorted queue by attribute.

IPOENT: Pointer to the entity (I) (integer*4)

IPOQUE: Pointer to the queue (I) (integer*4)

IATRIB: Attribute number for sort (I) (integer*4)

Subroutine CRIQUE (QUENAM, NQUELE, NREPRE, IONOFF, NUMECR, INDCRE, NULINE, NUCOLU, NNDHI1, NNDHI2, NLIMI1, NLIMI1, NLIMI2, NLIMSU, IPOINT)

- Creates a queue.

QUENAM: Queue name (I) (character*6)
NQUELE: Maximum number of queue elements (I) (integer*4)
IONOFF: ON/OFF flag (I) (integer*4)
NUMECR: Logical display number (I) (integer*4)
INDCRE: Growing direction (I) (integer*4)
NULINE: Logical display line number (I) (integer*4)
NUCOLU: Logical display column number (I) (integer*4)
NNDHI1: "Time in the queue" flag histogram (I) (integer*4)
NNDHI2: "Length in the queue" flag histogram (I) (integer*4)
NLIMI1: Last interval lower bound (hist. timei (I) (integer*4)
NLIMI2: Last interval lower bound (hist. length) (I) (integer*4)
NLIMSU: First interval upper bound (hist. time) (I) (integer*4)
IPOQUE: Queue pointer (O) (integer*4)

Function IDEQUE (IPOQUE, KPLACE)

- Retrieves the pointer to the entity currently at the KPLACEth position in a queue.

IPOQUE: Pointer to the queue (I) (integer*4)
KPLACE: Position in the queue (I) (integer*4)
IDEQUE: Entity pointer (O) (integer*4)

Function ISIZQU (IPOQUE)

- Retrieves the current number of elements in a queue.

IPOQUE: Pointer to the queue (I) (integer*4)
ISIZQU: Number of elements in the queue (O) (integer*4)

Subroutine TAKENT (IPOENT, IPOQUE)

- Takes an entity from a queue.

IPOENT: Pointer to the entity (I) (integer*4)
IPOQUE: Pointer to the queue (I) (integer*4)

Subroutine TAKFIR (IPOQUE)

- Takes an entity from the first position in a queue.

IPOQUE: Pointer to the queue (I) (integer*4)

Subroutine TAKLAS (IPOQUE)

- Takes an entity from the last position in a queue.

IPOQUE: Pointer to the queue (I) (integer*4)

Subroutine TAKPOS (IPOQUE, KPLACE)

- Takes an entity from the KPLACEth in a queue.

IPOQUE: Pointer to the queue (I) (integer*4)

KPLACE: Position in the queue (I) (integer*4)

TIME MANAGEMENT

Subroutine ADDTIME ()

- Adds one unit of time to the system clock.

Function NSTIME ()

- Retrieves the current internal simulation time.

NSTIME: Current simulation time (O) (integer*4)

VECTOR MANAGEMENT

Subroutine CREVEC (NDIMEN, IPOFRE)

- Reserve space and returns the pointer for a vector with size NDIMEN.

NDIMEN: Vector dimension (I) (integer*4)

IPOFRE: Vector pointer (O) (integer*4)

APPENDIX V.E

MSC Interactive Screen Generator Commands Description

(N) field creation

- Creates a field for display and/or read data within an application using the INSTA library subroutines. Optionaly it is possible to define text on the left of the data field.

The data field definition sequence is:

. use the arrow keys to position the cursor in the desired position;
. define text attributes (background colour, foreground colour, underline, character size);
. optional text input (full interactive editing facilities);
. define the mask attributes (position, mask character, mask character foreground colour, mask background colour, mask foreground colour, underline);
. define the field type (character fixed or variable length, integer fixed or variable length, real fixed or variable length);
. define the numeric data field limits (lower and upper bound values; the validation is made automatically by the INSTA screen management subroutines);
. define the fill option (required, default value, optional, display only).

(T) text creation

- Creates text on the screen as titles, comments or menu options. The text definition sequence is:

. use the arrow keys to position the cursor in the desired position;
. define text attributes (background colour, foreground colour, underline, character size);
. text input (full interactive editing facilities).

(A) erase

- Deletes a data field or text on the screen. The deletion sequence is:

. use the arrow keys to position the cursor at the beginning of the data field or text to be deleted (the TAB key moves the cursor to the beginning of the next data field or text; the HOME key moves the cursor to the beginning of the previous data field or text);
. press the A key to delete.

(Esc) end

- Quit the utility and let the user save the screen data to a file.

(P) position

- display at the bottom right of the screen the line and column coordinates for the current cursor position.

(R) repeat

- Repeats a data field or text on the screen. The repetition sequence is:
  - . position the cursor at the beginning of the data or text field to be repeated;
  - . press the R key;
  - . define the number of times to repeat up;
  - . define the line increment up if applicable;
  - . define the number of times to repeat left;
  - . define the column increment left if applicable;
  - . define the number of times to repeat right;
  - . define the column increment right if applicable;
  - . define the number of times to repeat down;
  - . define the line increment down if applicable;

(V) variable repeat

- Repeats a data field a number of times specified during the running of the user application. The variable repeat procedure is:
  - . position the cursor at the beginning of the data field to be repeated;
  - . press the V key;
  - . define the number of columns to the right;
  - . define the column increment to the right;
  - . define the line increment down.

(D) move

- Moves a data field or text around the screen. The move sequence is:
  - . position the cursor at the beginning of the data or text field to be moved;
  - . press the D key;
  - . move the cursor to the beginning of the new position;
  - . press the RETURN key;

(E) edit

- Edits a data or text field. The edit command sequence is:
  - . position the cursor at the beginning of the data or text field to be edited;
  - . press the E key;
  - . change any of the parameters associated with the data or text field.

(I) information

- Displays on the screen all the information concerning a data field. The information command procedure is:
  - . position the cursor at the beginning of a data field (if the cursor is not at the beginning of a data field then the first data field will be used);
  - . press the I key;
  - . the information concerning the particular data field is displayed on the screen and the following commands are available:
    - . (Esc) quits from the information screen;
    - . (C) input a new data field number;
    - . (RETURN) goes to the next data field.

(L) screen erase

- define if the terminal screen is to be erased before displaying the screen data.
  - . press the L key;
  - . define the erase screen parameter (Y/N);
  - . choose the erase screen colour.

(J) join

- Joins an external data screen with the current one. The join command sequence is:
  - . press the J key;
  - . defines the data screen file name to be joined with the current one.

(S) subroutine

- Generate a file with the FORTRAN 77 code to display and manage the current screen.

(C) O.S. command

- Let an operating system command to be executed.

(G) save

- save the current data screen to file.

APPENDIX V.F
AWARD Simulation Elements

ENTITIES

Nº	ENTITY	NAME	Num Att.	Alph. Att.
1	END_OF_SIMULATION_ENTITY	ENDSIM	1	0
2	INTERACTION_EVENT_ENTITY	INTEVE	1	0
3	SYSTEM_ENTITY	SYSTEM	100	5
4	CALENDAR_ENTITY	CAL	21	14
5	MONTH_ENTITY	MES	1	0
6	DAY_ENTITY	DIA	1	0
7	HOUR_ENTITY	HOR	1	0
8	MINUTE_ENTITY	MIN	1	0
9	REPORT_ENTITY	REPORT	10	0
10	GRAPHICAL_INFORMATION_ENTITY	GRAINF	25	0
11	JOB_TYPE_ENTITY	JOBTY1	3	0
12	TRANSPORTER_TYPE_ENTITY	TRTY11	21	1
13	TRANSPORTER_ENTITY	TR1122	56	0
14	VIRTUAL_TRANSPORTER_ENTITY	FR1122	4	0
15	TRANSPORTER_PATH_ENTITY	TP0001	11	0
16	JOB_ENTITY	TJ0001	7	0
17	OUTORDER_ENTITY	ORD001	11	0
18	OUTORDER_CONTROL_ENTITY	ORDCON	12	0
19	DESPATCH_BAY_ENTITY	DIS001	21	0
20	VIRTUAL_DESPATCH_BAY_ENTITY	FIS001	4	0
21	PALLET_ENTITY	PALLE1	1	0
22	CASE_ENTITY	CASES1	2	0
23	INORDER_ENTITY	INO001	9	0
24	INORDER_CONTROL_ENTITY	INOCON	10	0
25	RECEPTION_BAY_ENTITY	REP001	21	0
26	VIRTUAL_RECEPTION_BAY_ENTITY	FEP001	4	0
27	RACKING_ENTITY	RAC001	27	0
28	ZONE_ENTITY	ZON001	5	1
29	CELL_ENTITY	C01001	12	0
30	PRODUCT_GROUP_ENTITY	PG0001	14	1
31	PRODUCT_ENTITY	PR0011	13	0
32	TRANSPORTER_MOVEMENT_ENTITY	TRAMOV	16	0
33	NARROW_AISLE_ENTITY	NA0001	4	0

ENTITY ATTRIBUTES

1 - END_OF_SIMULATION_ENTITY

Maintains the end of simulation information.

Nº	Description
1	Maximum simulation time.

2 - INTERACTION_EVENT_ENTITY

Maintains the interactive event information.

Nº	Description
1	Interaction event flag. (0 = Off; 1 = On)

3 - SYSTEM_ENTITY

Maintains the system information.

Nº	Description
1	Movie Flag.
2	Transporter display increment.
3	Pointer to the GRAPHICAL_INFORMATION_ENTITY
4	Pointer to the PATHS_NEUTRAL_QUEUE.
5	Pointer to the TRANSPORTER_MOVEMENT_ENTITY.
6	Current dump number.
7	Pointer to the OUTORDER_CONTROL_ENTITY.
8	Pointer to the DESPATCH_BAYS_WAITING_QUEUE.
9	Pointer to the DESPATCH_BAYS_PROCESSING_QUEUE.
10	Pointer to the OUTORDERS_NEUTRAL_QUEUE.
11	Pointer to the PRODUCT_GROUPS_NEUTRAL_QUEUE.
12	Pointer to the PALLETS_NEUTRAL_QUEUE.
13	Pointer to the OUTORDER_INTER_ARRIVAL_TIMES_VECTOR.
14	Pointer to the CASES_NEUTRAL_QUEUE.
15	Pointer to the JOB_TYPES_QUEUE.
16	Pointer to the JOBS_NEUTRAL_QUEUE.
17	(Free)
18	Number of product_groups.
19	Maximum number of products per product_group.
20	Pointer to the PRODUCTS_WAITING_FOR_REPLENISHMENT_QUEUE.
21	Pointer to the TRANSPORTER_TYPES_PROCESSING_QUEUE.
22	Pointer to the JOBS_PROCESSING_QUEUE.
23	Pointer to the TRANSPORTERS_MOVEMENT_QUEUE.
24	Pointer to the TRANSPORTERS_PROCESSING_QUEUE.
25	Pointer to the RECEPTION_BAYS_WAITING_QUEUE.
26	Total number of products.
27	Pointer to the INORDER_CONTROL_ENTITY.
28	Pointer to the RECEPTION_BAYS_PROCESSING_QUEUE.
29	Pointer to the INORDER_INTER_ARRIVAL_TIMES_VECTOR.
30	Pointer to the INORDERS_NEUTRAL_QUEUE.
31	Pointer to the INORDER_PRODUCTS_WAITING_QUEUE.

32	Pointer to the ZONES_QUEUE.
33	Pointer to the RECEPTION_BAYS_INACTIVE_QUEUE.
34	Pointer to the DESPATCH_BAYS_INACTIVE_QUEUE.
35	Pointer to the TRANSPORTERS_INACTIVE_QUEUE.
36	Pointer to the CALENDAR_ENTITY
37	Pointer to the REPORT_ENTITY.

ALPHANUMERIC ATTRIBUTES

Nº	Description
1	Application name.
2	Product file name.
3	Outorders file name.
4	Graphics file name.
5	

4 - CALENDAR_ENTITY

Maintains the calendar and time information.

Nº	Description
1	Days in January.
2	Days in February.
3	Days in March.
4	Days in April.
5	Days in May.
6	Days in June.
7	Days in July.
8	Days in August.
9	Days in September.
10	Days in October.
11	Days in November.
12	Days in December.
13	Current year.
14	Current month.
15	Current day.
16	Current hour.
17	Current minute.
18	Pointer to the MONTH_ENTITY.
19	Pointer to the DAY_ENTITY.
20	Pointer to the HOUR_ENTITY.
21	Pointer to the MINUTE_ENTITY.

ALPHANUMERIC ATTRIBUTES

Nº	Description
1	January.
2	February.
3	March.
4	April.
5	May.
6	June.
7	July.
8	August.

9	September.
10	October.
11	November.
12	December.
13	String of the actual date and time.
14	String of the initial date and time.

5 - MONTH_ENTITY

Maintains the month.

Nº	Description
1	Pointer to the CALENDAR_ENTITY.

6 - DAY_ENTITY

Maintains the day.

Nº	Description
1	Pointer to the CALENDAR_ENTITY.

7 - HOUR_ENTITY

Maintains the hour.

Nº	Description
1	Pointer to the CALENDAR_ENTITY.

8 - MINUTE_ENTITY

Maintains the minute.

Nº	Description
1	Pointer to the CALENDAR_ENTITY.

9 - REPORT_ENTITY

Maintains the report information.

Nº	Description
1	Report flag. (0 = Off; 1 = On)
2	Report time interval. (480 seconds)
3	
4	
5	
6	
7	
8	
9	
10	

10 - GRAPHICAL INFORMATION_ENTITY

Maintains the graphics screen information.

Nº	Description
1	View 1 Initial X viewport coordinate.
2	View 1 Initial Y viewport coordinate.
3	View 1 Final X viewport coordinate.
4	View 1 Final Y viewport coordinate.
5	View 2 Initial X viewport coordinate.
6	View 2 Initial Y viewport coordinate.
7	View 2 Final X viewport coordinate.
8	View 2 Final Y viewport coordinate.
9	View 1 Initial X window coordinate.
10	View 1 Initial Y window coordinate.
11	View 1 Final X window coordinate.
12	View 1 Final Y window coordinate.
13	View 2 Initial X window coordinate.
14	View 2 Initial Y window coordinate.
15	View 2 Final X window coordinate.
16	View 2 Final Y window coordinate.
17	Graphics scale.
18	X origin coordinate.
19	Y origin coordinate.
20	X end coordinate.
21	Y end coordinate.
22	X Screen resolution
23	Y Screen resolution
24	Shape factor

11 - JOB_TYPE_ENTITY

1	JOB_TYPE index. (1..MAXJOB)
2	Number of priority queues.
3	Pointer to the JOB_PRIORITY_QUEUE_POINTERS_VECTOR.

12 - TRANSPORTER_TYPE_ENTITY

Maintains information on a particular transporter_type.

Nº	Description
1	Fixed time for start and stop. Horizontal loaded. (Min.)
2	Speed. Horizontal loaded. (M/Min.)
3	Fixed time for start and stop. Horizontal unloaded. (Min.)
4	Speed. Horizontal unloaded. (M/Min.)
5	Fixed time for start and stop. Vertical up loaded. (Min.)
6	Speed. Vertical up loaded. (M/Min.)
7	Fixed time for start and stop. Vertical up unloaded. (Min.)
8	Speed. Vertical up unloaded. (M/Min.)
9	Fixed time for start and stop. Vertical down loaded. (Min.)
10	Speed. Vertical down loaded. (M/Min.)
11	Fixed time for start and stop. Vertical down unloaded. (Min.)
12	Speed. Vertical down unloaded. (M/Min.)

13	
14	Width. (M)
15	Length. (M)
16	Transporter_type index.
17	Number of cases picked per minute.
18	Pointer to the TRANSPORTER_TYPES_WAITING_QUEUE.
19	Transporter_type colour. (1..15)
20	Fixed time for unloading in the reception_bay.
21	Fixed time for unloading in the despatch_bay.

ALPHANUMERIC ATTRIBUTES

Nº	Description
1	Transporter type name.

13 - TRANSPORTER_ENTITY

Nº	Description
1	Transporter index. (1..MAXTRA)
2	Pointer to the TRANSPORTER_TYPE_ENTITY.
3	Pointer to the PALLET_ENTITY being transported.
4	Current X coordinate.
5	Current Y coordinate.
6	Current Z coordinate.
7	Direction angle. (Degrees)
8	Index of the current branch.
9	Occupation flag (-2= Idle returning to park; -1= Idle; 0= breakdown; 1-4= Job 1-4)
10	Pointer to the PATHS_QUEUE.
11	Pointer to the JOBS_WAITING_QUEUE.
12	Next X coordinate.
13	Next Y coordinate.
14	Current length moved along the TRANSPORTER_PATH.
15	Half width. (M)
16	Half length. (M)
17	Half cross. (M)
18	X increment. (M)
19	Y increment. (M)
20	Path increment. (M)
21	Path length.
22	Wait flag.
23	Next event to be scheduled with the TRANSPORTER_ENTITY.
24	Segment number.
25	Final path X coordinate.
26	Final path Y coordinate.
27	Park X coordinate.
28	Park Y coordinate.
29	Park branch index.
30	TRANSPORTER_TYPE index.
31	Pointer to the congested TRANSPORTER_ENTITY.
32	Pointer to the DESPATCH_BAY_ENTITY being processed.
33	Pointer to the transporter movement queue.
34	Initial operation time.
35	Time spent doing JOB 1. (UNLOADING) (seconds) (Statistical information.)
36	Time spent doing JO3 2. (LOADING) (seconds) (Statistical information.)

37	Time spent doing JOB 3. (REPLENISHMENT) (seconds)
38	Waiting time.
39	Idle time.
40	Number of movement branches.
41	Pointer to the MOVEMENT_BRANCH_INDICES_VECTOR.
42	Number of different jobs possible.
43	Pointer to the ACTIVE_JOB_INDICES_VECTOR.
44	Pointer to the PRIORITY_JOB_QUEUES_VECTOR.
45	Time spent doing JOB 4. (PICKING)
46	Time spent doing JOB 6. (BREAKDOWN)
47	Breakdown flag. (0 = working; 1 = Breakdown)
48	Activity flag. (0 = Out of service; 1 = In service)
49	Pointer to the VIRTUAL_TRANSPORTER_ENTITY.
50	Initial activity time.
51	
52	
53	Pointer to the TRANSPORTER_ENTITY waiting to get into the narrow_aisle.
54	Time at which the transporter started waiting to get into the narrow_aisle.
55	Pointer to the JOB_ENTITY being processed.
56	Index of the previous branch.

14 - VIRTUAL_TRANSPORTER_ENTITY

Used to schedule changes in availability of the transporter.

Nº	Description
1	Pointer to the associated TRANSPORTER_ENTITY.
2	Number of transporter shift_times.
3	Pointer to the TRANSPORTER_SHIFT_TIMES_VECTOR.
4	Index of the next transporter shift time.

15 - TRANSPORTER_PATH_ENTITY

Maintains information on the path between two nodes.

Nº	Description
1	Initial X coordinate.
2	Initial Y coordinate.
3	Final X coordinate.
4	Final Y coordinate.
5	X direction cosine.
6	Y direction cosine.
7	Velocity. (M/min)
8	Path length. (M)
9	Direction angle.
10	Branch number.
11	Pointer to the NARROW_AISLE_ENTITY.

16 - JOB_ENTITY

Maintains information on the jobs beings undertaken by the transporters.

Nº	Description
1	Job index.
2	Pointer to the TRANSPORTER_ENTITY.
3	Pointer to the DESPATCH_BAY_ENTITY. or:
	Pointer to the RECEPTION_BAY_ENTITY. or:
	Pointer to the reserve CELL_ENTITY. (Job 3)
4	Index of the current pick job. or:
	Pointer to the picking CELL_ENTITY. (Job 3) or:
	Pointer to the reserve CELL_ENTITY. (Job 2)
5	Number of pick jobs.
6	Pointer to the TRANSPORTER_CELL_ENTITIES_POINTERS_VECTOR.
7	Pointer to the TRANSPORTER_CASE_ENTITIES_TO_PICK_VECTOR.

17 - OUTORDER_ENTITY

Nº	Description
1	OUTORDER index.
2	Time of arrival.
3	Processing start time.
4	Processing finish time.
5	Pointer to the DESPATCH_BAY_ENTITY.
6	Total number of PALLETs.
7	Total number of cases.
8	Number of PALLETs out of stock.
9	Number of cases out of stock.
10	Number of PALLETs yet to satisfy.
11	Number of cases yet to satisfy.

18 - OUTORDER_CONTROL_ENTITY

Nº	Description
1	Index of the first OUTORDER waiting for a DESPATCH_BAY.
2	Time of arrival of the first OUTORDER waiting for a DESPATCH_BAY.
3	Index of the next OUTORDER to arrive.
4	Number of OUTORDERs waiting for a DESPATCH_BAY.
5	Total number of OUTORDERs.
6	Total number of OUTORDERs so far.
7	Total number of PALLETs output.
8	Total number of cases output.
9	Total number of PALLETs out of stock.
10	Total number of cases out of stock.
11	Total processing time. (Seconds.)
12	Total waiting time. (Seconds.)

19 - DESPATCH_BAY_ENTITY

Nº	Description
1	DESPATCH_BAY index. (1..MAXDEB)
2	X coordinate.

3	Y coordinate.
4	Index of the branch for the associated aisle.
5	Pointer to the OUTORDER_ENTITY being processed. (0 if in a neutral queue)
6	Pointer to the OUTORDER_PALLETS_WAITING_QUEUE.
7	Pointer to the OUTORDER_CASES_WAITING_QUEUE.
8	Pointer to the TRANSPORTER_ENTITY for pallets.
9	Pointer to the TRANSPORTER_ENTITY for cases.
10	Truck segment number.
11	Angle.
12	Pivot X coordinate.
13	Pivot Y coordinate.
14	Box X1 coordinate.
15	Box Y1 coordinate.
16	Box X2 coordinate.
17	Box Y2 coordinate.
18	Despatch area colour.
19	Activity flag. (0 = out of service; 1 = in service)
20	Type of BAY. (Despatch = 1.0, Reception = 0.0)
21	Pointer to the VIRTUAL_DESPATCH_BAY_ENTITY.

20 - VIRTUAL_DESPATCH_BAY_ENTITY

Used to schedule changes in availability of the DESPATCH_BAY.

Nº	Description
1	Pointer to the associated DESPATCH_BAY_ENTITY.
2	Number of despatch_bay shift_times.
3	Pointer to the DESPATCH_BAY_SHIFT_TIMES_VECTOR.
4	Index of the next shift time.

21 - PALLET_ENTITY

Nº	Description
1	Pointer to the CELL_ENTITY.

22 - CASE_ENTITY

Maintains information on the cases to be picked.

Nº	Description
1	Pointer to the CELL_ENTITY.
2	Number of cases.

23 - INORDER_ENTITY

Nº	Description
1	INORDER index. (1..MAXINO)
2	Time of arrival.
3	Time of processing start.
4	Time of processing end.
5	Pointer to the RECEPTION_BAY_ENTITY.
6	Total number of PALLETs.

7	Number of PALLETs waiting for cells.
8	Number of PALLETs yet to satisfy.
9	Pointer to the PRODUCT_ENTITY.

24 - INORDER_CONTROL_ENTITY

Nº	Description
1	Index of the first INORDER waiting for a RECEPTION_BAY
2	Time of arrival of the first INORDER waiting for a RECEPTION_BAY
3	Index of the next INORDER to arrive.
4	Number of INORDERs waiting for a RECEPTION_BAY.
5	Total number of INORDERs.
6	Time of arrival of the last INORDER waiting for a RECEPTION_BAY.
7	Number of INORDERs. (Size of the INORDERs vector.)
8	Total pallets input. (Statistical information.)
9	Total processing time.
10	

25 - RECEPTION_BAY_ENTITY

Nº	Description
1	RECEPTION_BAY index. (1..MAXREB)
2	X coordinate.
3	Y coordinate.
4	Branch number.
5	Pointer to the INORDER_ENTITY being processed. (0 if in a neutral queue.)
6	Pointer to the INORDER_PALLETS_WAITING_QUEUE.
7	Pointer to the TRANSPORTER_ENTITY.
8	Pointer to the PALLET_ENTITY waiting for a vacant slot.
9	Number of the activity mark.
10	Truck segment number.
11	Angle.
12	Pivot X coordinate.
13	Pivot Y coordinate.
14	Box X1 coordinate.
15	Box Y1 coordinate.
16	Box X2 coordinate.
17	Box Y2 coordinate.
18	Reception bay colour.
19	Activity flag. (0 = Out of service; 1 = In service).
20	Type of bay. (Reception = 0.0; Despatch = 1.0).
21	Pointer to the VIRTUAL_RECEPTION_BAY_ENTITY.

26 - VIRTUAL_RECEPTION_BAY_ENTITY

Used to schedule changes in availability of the RECEPTION_BAY.

Nº	Description
1	Pointer to the associated RECEPTION_BAY_ENTITY.
2	Number of reception_bay shift_times.
3	Pointer to the RECEPTION_BAY_SHIFT_TIMES_VECTOR.
4	Index of the next shift time.

27 - RACKING_ENTITY

Nº	Description
1	RACKING_LOGIC_FLAG.
2	First cell X coordinate.
3	First cell Y coordinate.
4	X direction 1 cosine.
5	Y direction 1 cosine.
6	X direction 2 cosine.
7	Y direction 2 cosine.
8	Number of cells along direction 1.
9	Number of cells along direction 2.
10	cell width (M).
11	cell depth (M).
12	cell height (M).
13	Number of the first floor.
14	Number of the last floor.
15	Access direction flag. (1 or -1).
16	Pointer to the RACKING_ACCESS_INPUT_CELL_INDICES_VECTOR.
17	Pointer to the RACKING_ACCESS_INPUT_BRANCH_INDICES_VECTOR.
18	Pointer to the RACKING_ACCESS_OUTPUT_CELL_INDICES_VECTOR.
19	Pointer to the RACKING_ACCESS_OUTPUT_BRANCH_INDICES_VECTOR.
20	X lower of the access aisle at the beginning of the RACKING.
21	Y lower of the access aisle at the beginning of the RACKING.
22	X upper of the access aisle at the beginning of the RACKING.
23	Y upper of the access aisle at the beginning of the RACKING.
24	Mean access time for PMR's.
25	Number of transporters waiting for the PMR RACKING.
26	Size of the CELL_ENTITY_POINTERS_VECTOR.
27	Pointer to the PMR_WAITING_TRANSPORTERS_VECTOR.

28 - ZONE_ENTITY

Nº	Description
1	ZONE index.
2	Pointer to the ZONE_FREE_CELLS_QUEUE.
3	Number of slots in the ZONE.
4	Number of occupied SLOTs in the ZONE.
5	ZONE colour. (1..15)

29 - CELL_ENTITY

Nº	Description
1	cell index. (1..MAXCEL)
2	Pointer to the RACKING_ENTITY.
3	Number of SLOTs in the cell.
4	Pointer to the PRODUCT_ENTITY.
5	Number of occupied SLOTs in the cell.
6	SORT_KEY number.
7	Number of cases in the cell.
8	Number of cases ordered but not yet picked. (Reserved for outorders.)

9	Number of PALLETs ordered but not yet picked. (Reserved for outorders.)
10	Number of PALLETs received but not yet replenished.
11	Initial segment number.
12	Pointer to the ZONE_ENTITY.

30 - PRODUCT_GROUP_ENTITY

Nº	Description
1	Number of products in the product_group.
2	Average number of cases per SLOT.
3	Standard deviation of the number of cases per SLOT.
4	Maximum number of reserve PALLETs.
5	Minimum number of reserve PALLETs.
6	Pointer to the first preferred reserve storage ZONE_ENTITY.
7	Pointer to the second preferred reserve storage ZONE_ENTITY.
8	Pointer to the third preferred reserve storage ZONE_ENTITY.
9	Pointer to the picking ZONE_ENTITY.
10	Proportion of PALLET re-order level.
11	Pointer to the PRODUCTS_NEUTRAL_QUEUE.
12	Colour number for display. (1..15)
13	Number of pallets in each INORDER.
14	Time delay for INORDER arrival (hours).

31 - PRODUCT_ENTITY

Nº	Description
1	Pointer to the PRODUCT_GROUP_ENTITY.
2	Number of cases per pallet.
3	Pointer to the picking CELL_ENTITY.
4	Number of cell pointers. (cells containing the product.)
5	Pointer to the CELL_ENTITY_POINTERS_VECTOR.
6	Number of PALLET stock-out conditions.
7	Number of PALLETs out of stock.
8	Number of case stock-out conditions.
9	Number of cases out of stock.
10	Number of PALLETs.
11	REPLENISHMENT_FLAG. (0 = No; 1 = Yes)
12	INORDERING_FLAG. (0 = No; 1 = Yes; 2 = Processing)
13	Time of arrival of the INORDER.

32 - TRANSPORTER_MOVEMENT_ENTITY

Nº	Description
1	Number of movement net nodes.
2	Number of net branches.
3	Size of the ADJACENT_NODE_INDICES_VECTOR.
4	Pointer to the NODE_X_COORDINATES_VECTOR.
5	Pointer to the NODE_Y_COORDINATES_VECTOR.
6	Pointer to the BRANCH_LEFT_NODES_VECTOR.
7	Pointer to the BRANCH_RIGHT_NODES_VECTOR.
8	Pointer to the BRANCH_LENGTHS_VECTOR.
9	Pointer to the ADJACENT_NODE_INDICES_VECTOR.

10	Pointer to the ADJACENT_NODE_BRANCH_INDICES_VECTOR.
11	Pointer to the ADJACENT_NODE_POINTERS_VECTOR.
12	Pointer to the NARROW_AISLE_FLAG_VECTOR.
13	Pointer to the NARROW_AISLE_ENTITY pointer.

33 - NARROW_AISLE_ENTITY

Nº	Description
1	OCCUPATION_FLAG.
2	Pointer to the first TRANSPORTER_ENTITY waiting to get into the aisle.
3	Number of transporters in the aisle.

QUEUES

Nº	Queue Description	Queue Name
1	TRANSPORTER_TYPES_PROCESSING_QUEUE	TRTYQU
2	TRANSPORTERS_INACTIVE_QUEUE	QTRINA
3	JOB_TYPES_QUEUE	JOBTQU
4	JOB_PRIORITIES_QUEUE	QUJP12
5	TRANSPORTER_TYPES_WAITING_QUEUE	QUTY11
6	JOBS_WAITING_QUEUE	QJ1122
7	PATHS_QUEUE	QP1122
8	PATHS_NEUTRAL_QUEUE	QWTRAP
9	JOBS_NEUTRAL_QUEUE	QNTJOC
10	JOBS_PROCESSING_QUEUE	QPTRAJ
11	TRANSPORTERS_MOVEMENT_QUEUE	QTRAMO
12	TRANSPORTERS_PROCESSING_QUEUE	QPTRAN
13	OUTORDERS_NEUTRAL_QUEUE	QOUTOR
14	DESPATCH_BAYS_WAITING_QUEUE	QNEDIS
15	DESPATCH_BAYS_INACTIVE_QUEUE	QINDIS
16	DESPATCH_BAYS_PROCESSING_QUEUE	QPRDIS
17	OUTORDER_PALLETS_WAITING_QUEUE	QPW111
18	OUTORDER_CASES_WAITING_QUEUE	QCW111
19	PALLETS_NEUTRAL_QUEUE	QUENPA
20	CASES_NEUTRAL_QUEUE	QUENCA
21	INORDERS_NEUTRAL_QUEUE	QINORD
22	RECEPTION_BAYS_WAITING_QUEUE	QNEREP
23	RECEPTION_BAYS_INACTIVE_QUEUE	QINREP
24	RECEPTION_BAYS_PROCESSING_QUEUE	QPRECE
25	INORDER_PALLETS_WAITING_QUEUE	QPI111
26	RACKINGS_NEUTRAL_QUEUE	QUERAC
27	ZONES_QUEUE	QUEZON
28	ZONE_FREE_CELLS_QUEUE	QZO111
29	PRODUCT_GROUPS_NEUTRAL_QUEUE	QPRGRO
30	INORDER_PRODUCTS_WAITING_QUEUE	QWAPRO
31	PRODUCTS_WAITING_FOR_REPLENISHMENT_QUEUE	QWAPRE
32	PRODUCTS_NEUTRAL_QUEUE	PQ1111

VECTORS

Nº	Vector Name	Related entity
1	OUTORDER_INTER_ARRIVAL_TIMES	SYSTEM
2	INORDER_INTER_ARRIVAL_TIMES	SYSTEM
3	JOB_PRIORITY_QUEUE_POINTERS	JOB_TYPE
4	MOVEMENT_BRANCH_INDICES	TRANSPORTER
5	ACTIVE_JOB_INDICES	TRANSPORTER
6	PRIORITY_JOB_QUEUES	TRANSPORTER
7	TRANSPORTER_SHIFT_TIMES	VIRTUAL_TRANSPORTER
8	TRANSPORTER_CELL_ENTITIES_POINTERS	JOB
9	TRANSPORTER_CASE_ENTITIES_TO_PICK	JOB
10	DESPATCH_BAY_SHIFT_TIMES	VIRTUAL_DESPATCH_BAY
11	RECEPTION_BAY_SHIFT_TIMES	VIRTUAL_RECEPTION_BAY
12	RACKING_ACCESS_INPUT_CELL_INDICES	RACKING
13	RACKING_ACCESS_INPUT_BRANCH_INDICES	RACKING
14	RACKING_ACCESS_OUTPUT_CELL_INDICES	RACKING
15	RACKING_ACCESS_OUTPUT_BRANCH_INDICES	RACKING
16	PMR_WAITING_TRANSPORTERS	RACKING
17	CELL_ENTITY_POINTERS	PRODUCT
18	NODE_X_COORDINATES	TRANSPORTER_MOVEMENT
19	NODE_Y_COORDINATES	TRANSPORTER_MOVEMENT
20	BRANCH_LEFT_NODES	TRANSPORTER_MOVEMENT
21	BRANCH_RIGHT_NODES	TRANSPORTER_MOVEMENT
22	BRANCH_LENGTHS	TRANSPORTER_MOVEMENT
23	ADJACENT_NODE_INDICES	TRANSPORTER_MOVEMENT
24	ADJACENT_NODE_BRANCH_INDICES	TRANSPORTER_MOVEMENT
25	ADJACENT_NODE_POINTERS	TRANSPORTER_MOVEMENT
26	NARROW_AISLE_FLAG	TRANSPORTER_MOVEMENT

EVENTS

Nº	Associated entity	Description
1	END_OF_SIMULATION	End simulation.
2	HOUR	Change hour.
3	DAY	Change day.
4	MONTH	Change month.
5	MINUTE	Change minute.
6		(free)
7	TRANSPORTER	Free transporter.
8	JOB	Start a transporter job and update the transporter statistics.
9	JOB	Move a transporter for one increment of simulation time.
10	OUTORDER_CONTROL	Arrival of an *outorder*.
11	DESPATCH_BAY	Outorder initialisation.
12	DPATRD, DCATRD	Start processing an outorder.
13	DESPATCH_BAY	Finish processing an outorder.
14	JOB	Start transporter job 2. (LOADING)
15	DEPATH	Load a pallet from reserve storage. (LOADING)
16	JOB	Move a pallet to a despatch bay. (LOADING)
17	DEPATH	Unload a pallet to a despatch bay. (LOADING)
18	JOB	Return a transporter to park. (PARK)
19	DEPATH	Park a transporter. (PARK)
20	JOB	Start transporter job 4. (PICKING)
21	DEPATH	Start picking cases. (PICKING)
22	JOB	Finish picking cases. (PICKING)
23	JOB	Move cases to a despatch bay. (PICKING)
24	DEPATH	Unload cases at a despatch bay. (PICKING)
25	JOB	Start transporter job 3. (REPLENISHMENT)
26	DEPATH	Load a pallet from reserve storage. (REPLENISHMENT)
27	JOB	Take a pallet to picking storage for replenishment. (REPL.)
28	DEPATH	Start replenishing from pallet. (REPLENISHMENT)
29	JOB	Finish replenishing from pallet. (REPLENISHMENT)
30	JOB	Inorder arrival.
31	RECEPTION_BAY	Inorder initialisation.
32	JOB	Start transporter job 1. (UNLOADING)
33	DEPATH	Load a pallet from a reception bay. (UNLOADING)
34	JOB	Move a pallet to a reserve storage location. (UNLOADING)
35	DEPATH	Start unloading a pallet to a reserve storage location. (UNL.)
36	JOB	Finish unloading a pallet to a reserve storage location.(UNL.)
37	INORDER_CONTROL	Inorder end.
38	JOB	Return transporter to park breakdown or at end of activity(BR.)
39	DEPATH	Transporter in the park breakdown or at end of activity. (BR.)
40	VIRTUAL_TRANSPORTER	Transporter shift starts.
41	VIRTUAL_TRANSPORTER	Transporter shift ends.
42	VIRTUAL_DESPATCH_BAY	Bay shift starts.
	VIRTUAL_RECEPTION_BAY	
43	VIRTUAL_DESPATCH_BAY	Bay shift ends.
	VIRTUAL_RECEPTION_BAY	
44	REPORT	Generate a report and schedule the next one.